1era Ed.
2026

PUERTO MADERO EDITORIAL

INTRODUCCIÓN A LA PROGRAMACIÓN

Introduction to Programming

Teresita Lourdes Argüello Estrella
Patricia Isabel Albán Yanez
Enrique Marcelo Baño León
Fabian Eduardo Fierro Saltos
Galuth Irene García Camacho
Edwin Patricio Avilés Meza

puertomaderoeditorial.com.ar
La Plata - Argentina

ISBN: 978-631-6960-03-0

INTRODUCCIÓN A LA PROGRAMACIÓN

Introduction to Programming

AUTORES:
Teresita Lourdes Argüello Estrella
Fabian Eduardo Fierro Saltos
Enrique Marcelo Baño León
Galuth Irene García Camacho
Patricia Isabel Albán Yanez
Edwin Patricio Avilés Meza

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

INTRODUCCIÓN A LA PROGRAMACIÓN

Introduction to Programming

TOMO I

Autores:
Teresita Lourdes Argüello Estrella
Fabian Eduardo Fierro Saltos
Enrique Marcelo Baño León
Galuth Irene García Camacho
Patricia Isabel Albán Yanez
Edwin Patricio Avilés Meza

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

Introducción a la Programación / Teresita Lourdes Argüello Estrella ... [et al.] ;
Editado por
Juan Carlos Santillán Lima ; Sandra Isabel González Polo. - 1a ed. revisada. -
La Plata :
Puerto Madero Editorial Académica, 2026.
Libro digital, PDF/A

Archivo Digital: descarga y online
ISBN 978-631-6960-03-0

1. Lenguaje de Programación. I. Argüello Estrella, Teresita Lourdes II. Santillán Lima, Juan Carlos, ed. III. González Polo, Sandra Isabel, ed.
CDD 005

ISBN: 978-631-6960-03-0

1a ed. revisada

TÍTULO: Introducción a la Programación
Introduction to Programming
ISBN: 978-631-6960-03-0
Publicación: Septiembre de 2026

Editado por:
Sello editorial: ©Puerto Madero Editorial Académica
N° de Alta: 933832

Editorial: ©Puerto Madero Editorial Académica
CUIL: 20630333971
Calle 45 N491 entre 4 y 5
Dirección de Publicaciones Científicas Puerto Madero Editorial Académica
La Plata, Buenos Aires, Argentina
Teléfono: +54 9 221 314 5902
+54 9 221 531 5142
Código Postal: AR1900

Este libro se sometió a arbitraje bajo el sistema de doble ciego (peer review)

Corrección y diseño:

Puerto Madero Editorial Académica
Diseñador Gráfico: José Luis Santillán Lima

Diseño, Montaje y Producción Editorial:

Puerto Madero Editorial Académica
Diseñador Gráfico: Santillán Lima, José Luis

Director editorial: Santillán Lima, Juan Carlos
Editado por: Santillán Lima, Juan Carlos
Editado por: González Polo, Sandra Isabel

Hecho en Argentina
Made in Argentina

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

AUTORES

Teresita Lourdes Argüello Estrella
Magisterio Nacional Ecuatoriano. 020101, Guaranda, Bolívar, Ecuador.
teresita.arguello@docentes.educacion.edu.ec
https://orcid.org/0009-0002-1428-6719

Fabian Eduardo Fierro Saltos
Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.
fabian.fierro@ueb.edu.ec
https://orcid.org/0009-0002-8698-5930

Enrique Marcelo Baño León
Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.
enrique.bano@ueb.edu.ec
https://orcid.org/0000-0002-5550-0649

Galuth Irene García Camacho
Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.
ggarcia@ueb.edu.ec
https://orcid.org/0000-0001-8692-4017

Patricia Isabel Albán Yanez
Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.
paalban@ueb.edu.ec
https://orcid.org/0000-0003-3799-4195

Edwin Patricio Avilés Meza
Universidad Estatal de Milagro. 091050, Milagro, Manabí, Ecuador.
eavilesm6@unemi.edu.ec
https://orcid.org/0009-0002-4767-3424

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

Índice general

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

Índice de figuras

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

ISBN: 978-631-6960-03-0

Índice de cuadros

ISBN: 978-631-6960-03-0

RESUMEN

Este libro tiene como objetivo principal ofrecer una introducción estructurada y pedagógica a los fundamentos de la programación, dirigida a estudiantes universitarios y autodidactas. Para ello, se adopta un método de enseñanza gradual que integra el desarrollo del pensamiento algorítmico, utilizando herramientas didácticas como PSeInt, con la posterior implementación práctica en un entorno profesional como Visual Studio. El contenido se organiza en doce capítulos que abarcan desde conceptos básicos, lógica y algoritmos, hasta temas avanzados como estructuras de datos, programación modular y manejo de archivos.

Como resultado, se presenta un manual integral que combina teoría sólida, ejemplos prácticos y una extensa colección de ejercicios propuestos y resueltos. Los capítulos incorporan diagramas, flujogramas y tablas comparativas que facilitan la comprensión visual y la aplicación de los conceptos. Se destaca la transición efectiva entre el diseño lógico en pseudocódigo y la codificación en un lenguaje real, superando una barrera común en el aprendizaje inicial.

La conclusión principal es que el enfoque propuesto, basado en la integración de herramientas y un recorrido progresivo desde la lógica hacia la implementación, constituye una metodología efectiva para cimentar las competencias fundamentales en programación, preparando al lector para abordar desafíos computacionales más complejos y diversos lenguajes de programación.

Palabras clave: programación introductoria, algoritmos, pensamiento lógico, PSeInt, Visual Studio, enseñanza gradual, estructuras de datos, ejercicios prácticos

ISBN: 978-631-6960-03-0

ABSTRACT

This book's main objective is to offer a structured and pedagogical introduction to the fundamentals of programming, aimed at university students and self-taught students. To do this, a gradual teaching method is adopted that integrates the development of algorithmic thinking, using teaching tools such as PSeInt, with the subsequent practical implementation in a professional environment such as Visual Studio. The content is organized into twelve chapters that range from basic concepts, logic and algorithms, to advanced topics such as data structures, modular programming and file management.

As a result, a comprehensive manual is presented that combines solid theory, practical examples and an extensive collection of proposed and solved exercises. The chapters incorporate diagrams, flowcharts and comparative tables that facilitate visual understanding and application of the concepts. The effective transition between logical design in pseudocode and coding in a real language is highlighted, overcoming a common barrier in initial learning.

The main conclusion is that the proposed approach, based on the integration of tools and a progressive journey from logic to implementation, constitutes an effective methodology to build fundamental skills in programming, preparing the reader to address more complex computational challenges and diverse programming languages.

Keywords: introductory programming, algorithms, logical thinking, PSeInt, Visual Studio, gradual teaching, data structures, practical exercises

ISBN: 978-631-6960-03-0

PRÓLOGO

La programación constituye hoy una competencia fundamental para comprender y transformar los entornos tecnológicos que forman parte de la vida académica y profesional. Aprender a programar, sin embargo, no significa únicamente conocer la sintaxis de un lenguaje; supone desarrollar una forma ordenada de pensar, analizar problemas, construir soluciones paso a paso y comprobar críticamente los resultados obtenidos.

En este contexto, *Introducción a la Programación* ofrece un recorrido progresivo por los fundamentos del pensamiento computacional y la construcción de programas. La obra inicia con conceptos esenciales sobre datos, información, lógica y desarrollo de software; continúa con algoritmos y diagramas de flujo; y avanza hacia tipos de datos, variables, operadores, estructuras de control, arreglos, cadenas, programación modular, métodos de ordenamiento, búsqueda y manejo de ficheros.

Un aspecto especialmente valioso es la articulación entre la comprensión lógica y la implementación práctica. El uso de PSeInt permite que el estudiante se concentre primero en la estructura de la solución y en el razonamiento algorítmico, para posteriormente trasladar esas ideas a un entorno de programación como Visual Studio. Esta transición gradual facilita la comprensión del vínculo entre el pseudocódigo y un programa ejecutable.

Los ejemplos, diagramas, tablas comparativas y ejercicios propuestos y resueltos refuerzan el carácter formativo del libro. El lector no se limita a revisar definiciones: puede aplicar los conceptos, reconocer patrones de solución y fortalecer progresivamente su autonomía frente a problemas de mayor complejidad. Por ello, la obra puede servir tanto a estudiantes que inician su formación como a docentes y lectores autodidactas que necesitan una base estructurada para revisar principios fundamentales de programación.

Desde Puerto Madero Editorial Académica consideramos que la producción de materiales académicos claros, rigurosos y aplicables aporta de manera directa a la formación universitaria. Esta obra responde a ese propósito al integrar fundamentos conceptuales, herramientas didácticas y práctica sistemática en un mismo itinerario de aprendizaje. Invito al lector a recorrer estas páginas de manera activa, reproducir los ejemplos, resolver los ejercicios y asumir la programación como un proceso de construcción lógica, creativa y verificable. Esperamos que este libro constituya una base sólida para continuar explorando nuevos lenguajes, paradigmas y desafíos dentro de las ciencias de la computación.

Juan Carlos Santillán Lima
Director editorial
Puerto Madero Editorial Académica

ISBN: 978-631-6960-03-0

INTRODUCCIÓN

La programación informática constituye una disciplina fundamental en la formación de profesionales del área tecnológica, ya que desarrolla el pensamiento lógico, algorítmico y estructurado necesario para resolver problemas mediante soluciones computacionales. Sin embargo, la enseñanza de la programación enfrenta desafíos significativos, especialmente en etapas iniciales, debido a la complejidad sintáctica de los lenguajes y la necesidad de comprender conceptos abstractos antes de implementar código ejecutable. Este libro, titulado Introducción a la Programación, surge como respuesta a dicha problemática, ofreciendo un enfoque pedagógico gradual que integra herramientas educativas y entornos profesionales de desarrollo.

La formulación del problema se centra en la dificultad que experimentan los estudiantes al enfrentarse simultáneamente a la lógica algorítmica y a la sintaxis de un lenguaje de programación, lo que puede generar frustración y deserción en los cursos introductorios. Además, existe una brecha entre el diseño de algoritmos y su implementación práctica, que limita la transferencia de conocimiento desde el pseudocódigo hacia programas funcionales.

Los objetivos de esta obra son:

- Presentar los conceptos fundamentales de la programación de manera estructurada y progresiva, desde nociones básicas hasta estructuras de datos avanzadas.
- Integrar herramientas didácticas como PSeInt para el desarrollo del pensamiento algorítmico, y entornos profesionales como Visual Studio para la codificación real.
- Proporcionar una amplia colección de ejercicios propuestos y resueltos que faciliten la práctica y consolidación de los conocimientos adquiridos.

La justificación de este trabajo radica en la necesidad de contar con un material didáctico en español que combine rigor teórico con un enfoque aplicado, dirigido a estudiantes universitarios y autodidactas que se inician en el mundo de la programación. Al estructurar el contenido en capítulos que avanzan desde la lógica booleana hasta el manejo de archivos, se busca construir una base sólida que permita al lector abordar posteriormente lenguajes específicos y paradigmas más complejos.

ISBN: 978-631-6960-03-0

La estructura del libro sigue un flujo natural: inicia con conceptos básicos y lógica computacional, prosigue con algoritmos y estructuras de control, introduce el manejo de datos mediante arreglos y cadenas, explora la programación modular y algoritmos de ordenamiento, y culmina con el manejo de archivos y ejercicios integradores. Cada capítulo incluye ejemplos, diagramas y tablas comparativas que facilitan la comprensión y aplicación de los contenidos.

En definitiva, esta obra busca llenar un vacío en la literatura introductoria en español, ofreciendo un recorrido completo, accesible y aplicado que prepare al lector para enfrentar retos de programación en contextos académicos y profesionales.

ISBN: 978-631-6960-03-0

CAPÍTULO 1

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

La programación es un proceso para convertir especificaciones generales de un sistema en instrucciones utilizables por la máquina, que produzcan los resultados deseados; por lo que se le conoce también como desarrollo de software (Joyanes Aguilar, 1996). Este capítulo introduce los conceptos fundamentales que todo aspirante a programador debe comprender, estableciendo las bases para el aprendizaje posterior.

1.1 Programa

Un programa es una lista estructurada de instrucciones que la computadora ejecuta para procesar datos y generar información útil. Sin programas, el hardware sería incapaz de realizar tareas específicas. En otras palabras, el programa es el componente lógico que da vida a la máquina.

1.1.1 Características del Programa

Para que un programa sea efectivo, debe cumplir con ciertas características esenciales que aseguren su correcto funcionamiento y mantenimiento.

Confiabilidad y funcionalidad

El programa debe realizar las tareas para las que fue diseñado de manera consistente y sin fallos inesperados. La confiabilidad se refiere a su estabilidad, mientras la funcionalidad asegura que cumple todos los requisitos establecidos.

Detección de errores de entrada

Un programa robusto debe validar los datos de entrada, detectando y manejando errores comunes. Por ejemplo, si se espera un número, debe rechazar caracteres alfabéticos y notificar al usuario apropiadamente.

ISBN: 978-631-6960-03-0

Documentación adecuada

La documentación incluye comentarios en el código, manuales de usuario y especificaciones técnicas. Esta característica facilita el mantenimiento y la comprensión del programa por otros desarrolladores (Leestma & Nyhoff, 1999).

Comprensibilidad

El código debe ser claro y legible, utilizando nombres de variables significativos y una estructura lógica. Un programa comprensible es más fácil de modificar y depurar cuando sea necesario.

Codificación en lenguaje apropiado

La elección del lenguaje de programación debe basarse en el tipo de problema a resolver, la plataforma objetivo y consideraciones de rendimiento. No existe un lenguaje universalmente superior para todas las aplicaciones.

1.2 Datos

Los datos son valores brutos y aislados que representan características de entidades o eventos. Constituyen la materia prima que el programa procesa. Por ejemplo, en un sistema de ventas, los datos incluyen precios, cantidades y fechas de transacciones. Según (Ureña López, 1997) los datos carecen de significado por sí mismos hasta que son procesados.

1.3 Información

La información es el conocimiento útil derivado del procesamiento de datos. Mientras los datos son elementos discretos, la información representa datos organizados con significado contextual. Por ejemplo, del procesamiento de datos de ventas se obtiene información como .^{el} producto X fue el más vendido en el trimestre".

ISBN: 978-631-6960-03-0

Tabla 1.1: Diferencias clave entre datos e información.

Aspecto	Datos	Información
Naturaleza	Valores brutos	Datos procesados
Significado	Aislado, sin contexto	Contextualizado, con significado
Utilidad	Materia prima	Base para decisiones
Ejemplo	25, Rosa", 90	Rosa tiene 25 años y promedio 90"

Elaboración propia.

1.4 Procesamiento de Datos

El procesamiento de datos es el conjunto de operaciones que transforman datos en información. Este proceso sigue una secuencia lógica que incluye desde la recolección inicial hasta la presentación final de resultados.

1.4.1 Actividades del Procesamiento de Datos

El procesamiento implica tres actividades fundamentales que ocurren de manera secuencial.

Captura de datos de entrada

Esta fase inicial consiste en recolectar y registrar los datos desde diversas fuentes. La precisión en esta etapa es crucial, ya que errores aquí se propagan a todo el proceso. La captura puede ser manual o automática, dependiendo del sistema.

Manejo de datos

Una vez capturados, los datos son organizados y transformados. Esto incluye operaciones como clasificación (agrupar por categorías), ordenamiento (disponer en secuencia), cálculo (operaciones aritméticas) y validación (verificar consistencia).

ISBN: 978-631-6960-03-0

Administración de la salida resultante

Finalmente, la información procesada se presenta en formatos útiles para los usuarios. Esto puede incluir reportes impresos, visualizaciones en pantalla, archivos electrónicos o respuestas automáticas a otros sistemas.

1.5 Pasos del Desarrollo de Software

El desarrollo de software es el proceso estructurado para crear programas, abarcando desde la concepción inicial hasta el mantenimiento continuo. Sigue un ciclo de vida que organiza el trabajo en fases específicas.

El ciclo tradicional incluye seis etapas principales que garantizan la creación de software de calidad (Cairó Battistutti, 2005).

- Especificación del programa
- Diseño del programa
- Codificación del programa
- Prueba
- Documentación
- Mantenimiento

1.5.1 Especificación del programa

Se conoce como definición del problema o análisis del programa. Aquí se determina la información inicial para la elaboración del programa. Se define que debe resolver con el computador, de que presupuestos se debe partir, en definitiva, es el planteamiento del problema. Se requieren cinco tareas:

a. *Determinación de objetivos del programa.*

Se define los problemas particulares que deberán ser resueltos o las tareas a realizar, esto permitirá saber qué es lo que se pretende solucionar y proporcionara información útil para el planeamiento de la solución.

ISBN: 978-631-6960-03-0

b. *Determinación de la salida deseada.*

Los datos seleccionados deben ser arreglados en una forma ordenada para producir información. La salida será impresa o en el monitor.

c. *Determinación de los datos de entrada.*

Identificada la salida que se desea, se determina los datos de entrada y la fuente de estos datos. Los datos deben ser recolectados y analizados.

d. *Determinación de los requerimientos de procesamiento.*

Aquí se definen las tareas de procesamiento que deben desempeñarse para que los datos de entrada se conviertan en una salida.

e. *Documentación de las especificaciones del programa.*

Es importante disponer de documentación permanente. Deben registrarse todos los datos necesarios para el procesamiento requerido. Esto conduce al siguiente paso del diseño del programa.

1.5.2 Diseño del programa

Es diseñar cualquier sistema nuevo o las aplicaciones que se requieren para satisfacer las necesidades. Esta actividad se debe dividir en:

- Operaciones de entrada/salida
- Cálculos
- Lógica/ comparación
- Almacenamiento/ consulta

Aquí se genera una solución con técnicas de programación como diseño descendente de programas, pseudocódigos, flujogramas y estructuras lógicas.

ISBN: 978-631-6960-03-0

1.5.3 Codificación del programa

Es la generación real del programa con un lenguaje de programación. En esta etapa se hace uso de la lógica que desarrolló en el paso del diseño del programa para efectivamente generar un programa. Se debe seleccionar el lenguaje apropiado para resolver el problema.

Prueba y depuración del programa

Depurar es ejecutar el programa y corregir las partes que no funcionan. En esta fase se comprueba el funcionamiento de cada programa y esto se hace con datos reales o ficticios. Cuando los programas están depurados, se prueban. Cuando los programas se depuran, se pueden encontrar los siguientes errores:

- Errores de sintaxis o de compilación
- Errores de ejecución
- Errores de lógica
- Errores de especificación.

a. ***Errores de sintaxis o de compilación***

Es una violación de las reglas del lenguaje de programación. Son fáciles de corregir, porque los detecta el compilador (posible error de escritura), el da información sobre el lugar donde esta y la naturaleza de cada uno de ellos mediante un mensaje de error.

b. ***Errores de Ejecución***

Se deben generalmente a operaciones no permitidas como dividir por cero, leer un dato no numérico en una variable numérica, exceder un rango de valores permitidos, etc. Se detectan porque se produce una parada anormal del programa durante su ejecución.

c. ***Errores de Lógica***

Corresponden a la obtención de resultados que no son correctos y la única manera de detectarlos es realizando suficientes pruebas del programa. Son difíciles de corregir, no solo por la dificultad de detectarlos, sino porque se deben a la propia concepción y diseño del programa.

ISBN: 978-631-6960-03-0

d. ***Errores de Especificación***

Es el peor tipo de error y el más difícil de corregir. Se deben a mal diseño del programa posiblemente por mala comunicación usuario programador y se detectan cuando ya se ha concluido el diseño e instalación del programa, lo cual puede implicar repetir gran parte del trabajo realizado.

1.5.4 Prueba

Consiste en verificar la funcionalidad del programa a través de varios métodos para detectar errores posibles.

Métodos de Prueba:

- Chequeo de escritorio
- Prueba manual de datos de muestra
- Intento de traducción
- Prueba de datos de muestra en la computadora
- Prueba por un grupo selecto de usuarios potenciales.

a. ***Chequeo de Escritorio:***

El programador corrige una impresión del programa. Revisa el listado línea por línea en busca de errores de sintaxis y lógica.

b. ***Prueba manual de datos de muestra:***

Se corre el programa en forma manual aplicando datos tanto correctos como incorrectos para comprobar que funciona correctamente.

c. ***Intento de Traducción:***

El programa corre en una computadora usando un programa traductor para convertirlo a lenguaje de máquina. Para ello debe estar sin errores de sintaxis, de lo contrario serán identificados por el programa de traducción.

ISBN: 978-631-6960-03-0

d. ***Prueba de datos de muestra en la computadora:***

Después del intento de traducción y corregidos los errores de sintaxis, se procede a buscar errores de lógica utilizando diferentes datos de muestra.

e. ***Prueba por un grupo selecto de usuarios potenciales:***

Se trata del paso final en la prueba de un programa (prueba beta). Usuarios potenciales ponen a prueba el programa y ofrecen retroalimentación.

1.5.5 Documentación del programa

Consiste en describir a nivel técnico los procedimientos relacionados con el programa y su modo de uso. Se debe documentar el programa para que sea más entendible. La documentación sirve para los:

- Usuarios (Digitadores)
- Operadores
- Programadores
- Analistas de sistemas

Los documentos que se elaboran son: Manual de Usuario y Manual del Analista.

Al usuario se elabora un manual de referencia para que aprenda a utilizar el programa. Esto se hace a través de capacitaciones y revisión de la documentación del manual de usuario.

El manual del usuario no está escrito a nivel técnico sino al de los distintos usuarios previstos y explica en detalle cómo usar el programa:

- Descripción de las tareas que realiza el programa
- Instrucciones necesarias para su instalación puesta en marcha y funcionamiento
- Recomendaciones de uso
- Menús de opciones
- Método de entrada y salida de datos

ISBN: 978-631-6960-03-0

- Mensajes de error
- Recuperación de errores, etc.

A los operadores para que sepan cómo responder a los mensajes de error que se presentan. Además, que se encarguen de darle soporte técnico al programa.

A los programadores para que recuerden aspectos de la elaboración del programa o para que otras personas puedan actualizar o modificar (darle mantenimiento). Por eso la documentación debe contener algoritmos y flujogramas de los módulos y las relaciones que se establecen entre ellos; listados del programa, corridas, descripción de variables que se emplean en cada módulo (comunes y locales); descripción de los ficheros de cada módulo y todo lo que sea de importancia para un programador.

A los analistas de sistemas que son las personas que deberán proporcionar toda la información al programador. Estos se encargan de hacer una investigación previa de cómo realizar el programa y documentar con las herramientas necesarias para que el programador pueda desarrollar el sistema en algún lenguaje de programación adecuado.

1.5.6 Mantenimiento del programa

Es el paso final del desarrollo del software. Alrededor del 75 % del costo total del ciclo de vida de un programa se destina al mantenimiento. El propósito del mantenimiento es garantizar que los programas en uso estén libres de errores de operación y sean eficientes y efectivos.

Este capítulo ha establecido los conceptos fundamentales de la programación. En primer lugar, se definió qué es un programa y sus características esenciales. Posteriormente, se distinguió entre datos e información, elementos centrales del procesamiento. Finalmente, se describió el ciclo de desarrollo de software, desde la especificación hasta el mantenimiento. Estos conceptos básicos constituyen la base indispensable para avanzar en el estudio de la programación, que se desarrollará en los capítulos siguientes con temas como algoritmos, estructuras de control y lenguajes de programación.

ISBN: 978-631-6960-03-0

Figura 1.1: Relación entre los componentes básicos de la programación

Elaboración propia.

ISBN: 978-631-6960-03-0

CAPÍTULO 2

LÓGICA COMO ASPECTO BÁSICO DE LA PROGRAMACIÓN

La lógica constituye el fundamento intelectual de la programación. Se trata de la disciplina que permite pensar de manera estructurada sobre soluciones alternativas y sus posibles resultados, facilitando la toma de decisiones inteligentes durante el desarrollo de software. En otras palabras, programar es, en gran medida, aplicar principios lógicos para resolver problemas de manera sistemática. Este capítulo explora los conceptos lógicos esenciales y su directa aplicación en la construcción de algoritmos.

2.1 Fundamentos de Lógica Computacional

La lógica computacional es la rama que adapta los principios del razonamiento formal al ámbito de las ciencias de la computación. Proporciona las herramientas simbólicas para analizar problemas, diseñar soluciones y verificar su corrección antes de la implementación en código.

2.1.1 Definición de Lógica

La lógica puede definirse como el estudio crítico del razonamiento, particularmente de los métodos y principios utilizados para distinguir entre argumentos correctos (válidos) y argumentos incorrectos (inválidos) (Copi, 1998). En programación, esto se traduce en la capacidad de construir secuencias de instrucciones que, partiendo de ciertas condiciones iniciales (premisas), lleven inevitablemente a los resultados deseados (conclusiones).

2.1.2 Lógica Deductiva

Se encarga de determinar la validez de los argumentos. Un razonamiento deductivo es válido si la conclusión se sigue necesariamente de las premisas. Por ejemplo, en programación, si establecemos la premisa "SI el usuario es mayor de edad Y tiene saldo suficiente, ENTONCES puede realizar la transacción", aplicamos lógica deductiva para

ISBN: 978-631-6960-03-0

tomar una decisión en el código.

2.1.3 Argumentos

Cuando el razonamiento se expresa con palabras, recibe el nombre de .ªrgumento". Un argumento es un grupo cualquiera de proposiciones o enunciados que forman premisas y conclusiones. Este puede constar de varias premisas, pero de una sola conclusión.

2.1.4 Premisas y Conclusiones

Las premisas de un argumento son proposiciones afirmadas como fundamento o razones para aceptar una conclusión. La conclusión es la proposición afirmada que se basa en las otras proposiciones o premisas. Una proposición puede ser premisa en un argumento y conclusión en otro. Por ejemplo: "Todos los hombres son mortales.ªparece como premisa en el siguiente argumento:

"Todos los hombres son mortales"

"Sócrates es un hombre"

"Por lo tanto, Sócrates es mortal".

Y como conclusión en el siguiente argumento:

"Todos los animales son mortales"

"Todos los hombres son animales"

"Luego, todos los hombres son mortales"

Expresiones como "por tanto", "por lo cual", "de ello se deduce", sirven para introducir la conclusión de un argumento, en tanto que "puesz "porque"se emplean para introducir las premisas.

Otro Ejemplo:

"Todos se aburren en la conferencia".

"Ninguno de los que se aburren presta atención".

"Por consiguiente, ninguno de los asistentes está prestando atención".

Hay dos condiciones que debe satisfacer un argumento para establecer la verdad de su conclusión: Debe ser válido y todas sus premisas deben ser verdaderas.

ISBN: 978-631-6960-03-0

2.2 Expresiones Booleanas y Tablas de Verdad

Las expresiones booleanas son afirmaciones que solo pueden tomar uno de dos valores: **Verdadero (V o 1)** o **Falso (F o 0).** Constituyen la base de la toma de decisiones en los programas. Para analizar sistemáticamente cómo el valor de una expresión compleja depende de sus componentes, se utilizan las tablas de verdad.

Figura 2.1: Tablas de Verdad de los Operadores Lógicos

X	Y	And
V	V	V
V	F	F
F	V	F
F	F	F

X	Y	OR
V	V	V
V	F	V
F	V	V
F	F	F

X	NOT (X)
V	F
F	V

Elaboración propia.

2.2.1 Enunciados Simples y Compuestos

Un enunciado simple es una proposición que no contiene otra como parte componente, por ejemplo: "La variable edad es mayor a 18". Un enunciado compuesto se forma al unir enunciados simples mediante conectores lógicos (operadores). Por ejemplo: "La variable edad es mayor a 18 Y la variable saldo es mayor a 0".

2.2.2 Operadores Lógicos Básicos

Los tres operadores lógicos fundamentales son la conjunción (AND), la disyunción (OR) y la negación (NOT). Son los bloques de construcción para expresiones condicionales complejas.

Conjunción (AND)

El operador AND entre dos proposiciones **p** y **q** da como resultado **Verdadero** solo si ambas proposiciones son verdaderas. En caso contrario, el resultado es **Falso**. Es equivalente a la intersección en teoría de conjuntos.

Sintaxis en pseudocódigo: **(edad > 18) Y (saldo > 0)**

ISBN: 978-631-6960-03-0

Disyunción (OR)

El operador OR entre **p** y **q** da como resultado **Verdadero** si al menos una de las proposiciones es verdadera. Solo es **Falso** cuando ambas son falsas. Es equivalente a la unión en teoría de conjuntos.

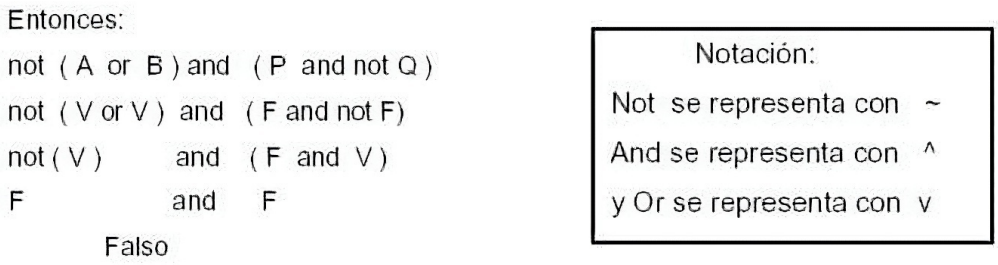
Sintaxis en pseudocódigo: **(nota >= 60) O (asistencia >= 80)**

Negación (NOT)

El operador NOT actúa sobre una sola proposición e invierte su valor de verdad. Si **p** es Verdadero, NO **p** es Falso, y viceversa.

Sintaxis en pseudocódigo: **NO (usuario_bloqueado)**

Figura 2.2: Notación de los Operadores Lógicos



Elaboración propia.

Ejemplos de aplicación de estas tablas de verdad:

Si A y B son valores verdaderos y P y Q son falsos, cuál es el valor de verdad de las siguientes expresiones:

not (A or B) and (P and not Q)

Solución: Evaluamos primero los paréntesis, comenzando con el primero. En la tabla OR buscamos a qué equivalen Verdadero or Verdadero (porque A y B son verdaderos según el enunciado). Obtenemos que es Verdadero. Como la expresión está negada, su valor opuesto es Falso. Al evaluar el segundo paréntesis, Q es falso y al negarlo nos queda verdadero. Luego, si P es falso nos queda Falso and Verdadero, y esto es igual a Falso.

Es de suma importancia que se comprenda las operaciones lógicas dado que no es suficiente por si sola; para que se garantice una expresión booleana esperada en un programa, además en la práctica, las condiciones pueden volverse complejas al combinar

ISBN: 978-631-6960-03-0

múltiples proposiciones, lo que puede incrementar el riesgo de errores lógicos que son difíciles de detectar únicamente con la intuición. Es así que, en este contexto, las tablas de verdad se convierten en una herramienta fundamental, dado que permite analizar de manera sistemática y exhaustiva todas las posibles combinaciones de valores de verdad de una expresión lógica. Ahora su uso facilita la correcta verificación del comportamiento de condiciones para que así contribuya en la construcción de programas confiables, y fáciles de depurar.

2.2.3 Construcción de Tablas de Verdad

Una tabla de verdad es una representación sistemática de todos los posibles valores de verdad de una expresión booleana. Para construirla:

1. Se listan todas las variables simples **(p, q, ...)** y se calculan todas las combinaciones posibles de **V** y **F**.
2. Se añaden columnas para cada subexpresión, calculando su valor según los operadores.
3. La columna final muestra el valor de la expresión completa para cada combinación.

Tabla 2.1: Tabla de verdad para los operadores básicos

p	q	AND p Y q	OR p O q	NOT NO p
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Elaboración propia.

ISBN: 978-631-6960-03-0

CAPÍTULO 3

LENGUAJES DE PROGRAMACIÓN: CONCEPTOS Y CLASIFICACIÓN

Un lenguaje de programación es un sistema de comunicación formal que permite a un programador escribir instrucciones que una computadora puede ejecutar. Estos lenguajes actúan como un puente esencial entre el pensamiento humano lógico y las operaciones binarias del hardware. Comprender su estructura, clasificación y funcionamiento es fundamental para seleccionar la herramienta adecuada para cada tarea de desarrollo de software.

3.1 Componentes de un Lenguaje

Todo lenguaje de programación se construye sobre tres componentes fundamentales que definen su forma y significado. Estos componentes trabajan en conjunto para permitir la creación de programas correctos y comprensibles.

3.1.1 Léxico

El léxico, o vocabulario, es el conjunto de símbolos básicos permitidos por el lenguaje. Incluye palabras reservadas (como si, mientras, entero), identificadores (nombres creados por el programador para variables y funciones), operadores (como **+, -, *, =)** y símbolos especiales (puntos, comas, paréntesis). Son las "palabrasçon las que se escribe el programa.

3.1.2 Sintaxis

La sintaxis define el conjunto de reglas que gobiernan cómo se pueden combinar los símbolos léxicos para formar instrucciones válidas. Es la "gramática"del lenguaje. Por ejemplo, la sintaxis dicta que una instrucción si debe ir seguida de una condición entre paréntesis. Un error de sintaxis, como olvidar un punto y coma en C++, hará que el programa no pueda ser traducido.

ISBN: 978-631-6960-03-0

3.1.3 Semántica

La semántica se refiere al significado de las construcciones sintácticas. Mientras la sintaxis dice cómo se escribe algo, la semántica define qué hace. Por ejemplo, la sintaxis permite escribir a = b + c;, y la semántica define que esta instrucción suma los valores de b y c y asigna el resultado a a. Un programa puede ser sintácticamente correcto, pero semánticamente erróneo si no hace lo que el programador pretendía.

3.2 Clasificación por Nivel

Los lenguajes de programación se clasifican según su proximidad al lenguaje de la máquina (bajo nivel) o al lenguaje humano (alto nivel). Esta proximidad implica un trade-off entre control del hardware y facilidad de programación.

3.2.1 Lenguajes de Bajo Nivel

Estos lenguajes están estrechamente vinculados a la arquitectura específica del procesador. El lenguaje de máquina (código binario de ceros y unos) es el único que el procesador entiende directamente. El lenguaje ensamblador utiliza códigos mnemotécnicos (como MOV o ADD) para representar instrucciones de máquina, ofreciendo un nivel de abstracción muy bajo. Su ventaja es el control total y la eficiencia máxima; su desventaja es la complejidad y falta de portabilidad.

3.2.2 Lenguajes de Alto Nivel

Estos lenguajes utilizan sintaxis y palabras cercanas al idioma humano (inglés) y a conceptos matemáticos. Ejemplos son Python, Java, C++ y JavaScript. Una sola instrucción en un lenguaje de alto nivel suele equivaler a varias instrucciones en lenguaje de máquina. Su principal ventaja es la productividad del programador, la legibilidad y la portabilidad (un programa puede ejecutarse en diferentes máquinas con un traductor adecuado).

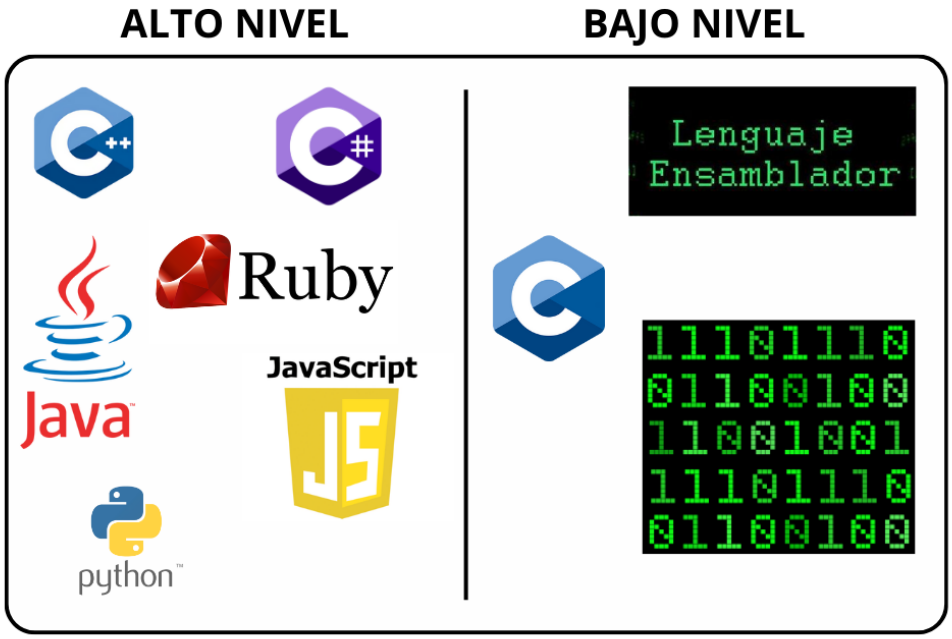
Características de los lenguajes de alto nivel:

- Son independientes de la arquitectura física del computador.

ISBN: 978-631-6960-03-0

- Permiten usar los mismos programas en computadores con diferentes arquitecturas (portabilidad), y no es necesario conocer el hardware específico de la máquina.
- La ejecución de un programa en lenguaje de alto nivel, requiere de una traducción al lenguaje del computador donde va a ser ejecutado.
- Una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje entendible por el computador.
- Utilizan notaciones cercanas a las usadas por las personas en un determinado ámbito.
- Se suelen incluir instrucciones potentes de uso frecuente que son ofrecidas por el lenguaje de programación.

Figura 3.1: Lenguajes de programación de bajo nivel y Alto nivel



Elaboración propia.

ISBN: 978-631-6960-03-0

3.3 Generaciones de Lenguajes

La evolución de los lenguajes de programación suele describirse en generaciones, que reflejan aumentos progresivos en el nivel de abstracción y la potencia expresiva.

- Lenguajes de máquina
- Lenguajes ensambladores
- Lenguajes de procedimientos
- Lenguajes orientados a problemas
- Lenguajes naturales

3.3.1 Primera Generación: Lenguaje de máquina

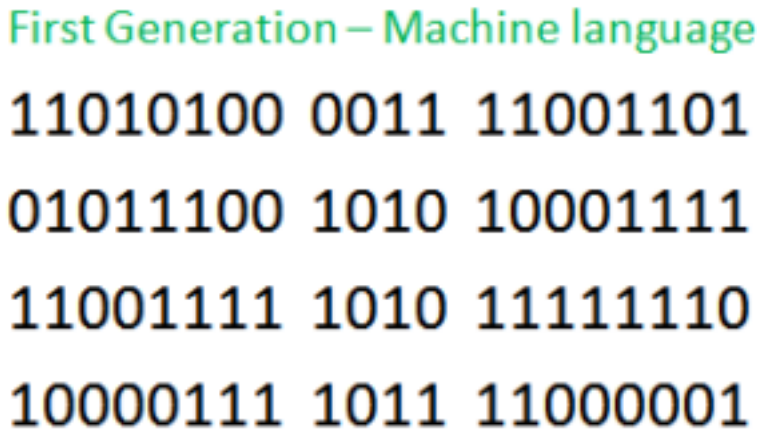
Es el lenguaje que el computador entiende, su estructura está adaptada a los circuitos de la máquina, la programación es tediosa porque los datos se representan por ceros y unos. Es de bajo nivel. Es un conjunto de instrucciones codificadas en binario que son capaces de relacionarse directamente con los registros y circuitería del microprocesador del computador y se ejecuta directamente sin necesidad de otros programas. Los datos se reverencian por medio de las direcciones de memoria donde se encuentran y las instrucciones realizan operaciones simples. Estos lenguajes están ligados a la CPU y por eso no son transferibles. (Baja portabilidad). Para los programadores es posible escribir programas directamente en lenguaje de máquina, pero las instrucciones son difíciles de recordar y los programas resultan largos y laboriosos de escribir y también de corregir y depurar.

3.3.2 Segunda Generación: Lenguaje ensamblador

Es otro lenguaje de programación de bajo nivel, pero simbólico porque las instrucciones se construyen usando códigos de tipo mnemotécnico, lo cual facilita la escritura y depuración de los programas, pero no los acorta puesto que para cada acción se necesita una instrucción. El programa ensamblador va traduciendo línea a línea a la vez que comprueba la existencia de errores. Si localiza alguno da un mensaje de error. Algunas características que lo diferencian del lenguaje de máquina son que permite el

ISBN: 978-631-6960-03-0

Figura 3.2: Lenguaje de la Maquina de la Primera Generación



Fuente: (Marketing, 2025)

uso de comentarios entre las líneas de instrucciones; en lugar de direcciones binarias usa identificadores como total, x, y, etc.

Y los códigos de operación se representan por mnemotécnica siempre tienen la desventaja de repertorio reducido de instrucciones, rígido formato para las instrucciones, baja portabilidad y fuerte dependencia del hardware. Tiene la ventaja del uso óptimo de los recursos hardware, permitiendo la obtención de un código muy eficiente. Ejemplo de algunos códigos mnemónicos son: STO para guardar un dato, LOA para cargar algo en el acumulador, ADD para adicionar un dato, INP para leer un dato, STO para guardar información, MOV para mover un dato y ponerlo en un registro, END para terminar el programa, etc. Con la tercera generación avanzamos a los lenguajes de alto nivel, muchos de los cuales se consideran exportables, se puede exportar de una máquina a otra.

3.3.3 Tercera Generación: Lenguajes de procedimientos

Marcaron un salto hacia la abstracción. Son lenguajes de alto nivel diseñados para expresar la lógica de procedimientos o algoritmos de forma más natural. Se centran en cómo se realiza una tarea mediante secuencias, decisiones y bucles. Introdujeron conceptos como variables, tipos de datos y subrutinas. Ejemplos: FORTRAN (cálculo científico), COBOL (negocios), C (sistemas), Pascal (educación). Fomentan la programación estructurada.

Son lenguajes de alto nivel similares al habla humana, pero requieren cierta

ISBN: 978-631-6960-03-0

Figura 3.3: Ejemplo del Lenguaje Ensamblador

Fuente: (Urbina, 2012)

ISBN: 978-631-6960-03-0

capacitación para su uso, se centran en cómo se realiza una tarea mediante secuencias, decisiones y bucles. Introdujeron conceptos como variables, tipos de datos y subrutinas.

Ventajas

1. Independencia de la arquitectura física de la computadora (portabilidad), esto significa que un mismo lenguaje puede funcionar (al menos en teoría) en distintos computadores, por lo que tanto el lenguaje como los programas escritos con él serán transportables de un computador a otro. En la práctica, esta característica resulta limitada por la gran diversidad de versiones y dialectos que se constituyen para cada lenguaje.

2. Una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje máquina. Se llaman de procedimientos porque están diseñados para expresar la lógica capaz de resolver problemas generales. Entre estos tenemos:

   FORTRAN (cálculo científico), COBOL (negocios), C (sistemas), Pascal (educación) y BASIC

Para que el lenguaje de procedimientos pueda funcionar debe traducirse a lenguaje de máquina a fin de que la computadora lo entienda. Para ello se han de usar programas traductores que realicen dicho proceso. Tienen la capacidad de soportar programación estructurada.

3.3.4 Cuarta Generación: Lenguajes orientados a problemas

Resultan más eficaces para la resolución de un tipo de problemas a costa de una menor eficiencia para otros. Requieren poca capacitación especial de parte del usuario Son considerados de muy alto nivel Diseñados para resolver problemas específicos

Incluye: lenguajes de consulta y generador de aplicaciones

Lenguajes de consulta: Permiten a no programadores usar ciertos comandos de fácil comprensión para la búsqueda y generación de reportes a partir de una base de datos Ejemplo: SQL.

Generador de aplicaciones: Quiere decir que cuando se diseña uno de estos lenguajes, se tiene en cuenta que su finalidad es la resolución de problemas, prescindiendo

ISBN: 978-631-6960-03-0

Figura 3.4: Ejemplo de Lenguaje C

```
#include <stdio.h>

int main()
{
   printf("Hola Mundo!\n");
   return 0;
}
```

Fuente: (Duglas, 2013)

de la arquitectura del computador. Contiene varios módulos que han sido preprogramados para cumplir varias tareas.

3.3.5 Quinta Generación: Lenguajes naturales e IA

Lenguajes orientados a aplicaciones en inteligencia artificial, como lisp y prolog.

Dentro de este campo destacan las aplicaciones en sistemas expertos, juegos, visión artificial (Jurasic Park) y robótica. Lisp es un lenguaje para procesamiento de listas y manipulación de símbolos. Prolog es un lenguaje basado en la lógica, para aplicaciones de bases de datos e Inteligencia Artificial.

Podemos decir entonces, que los lenguajes de alto nivel tienen las ventajas de mayor legibilidad de los programas, portabilidad, facilidad de aprendizaje y facilidad de modificación.

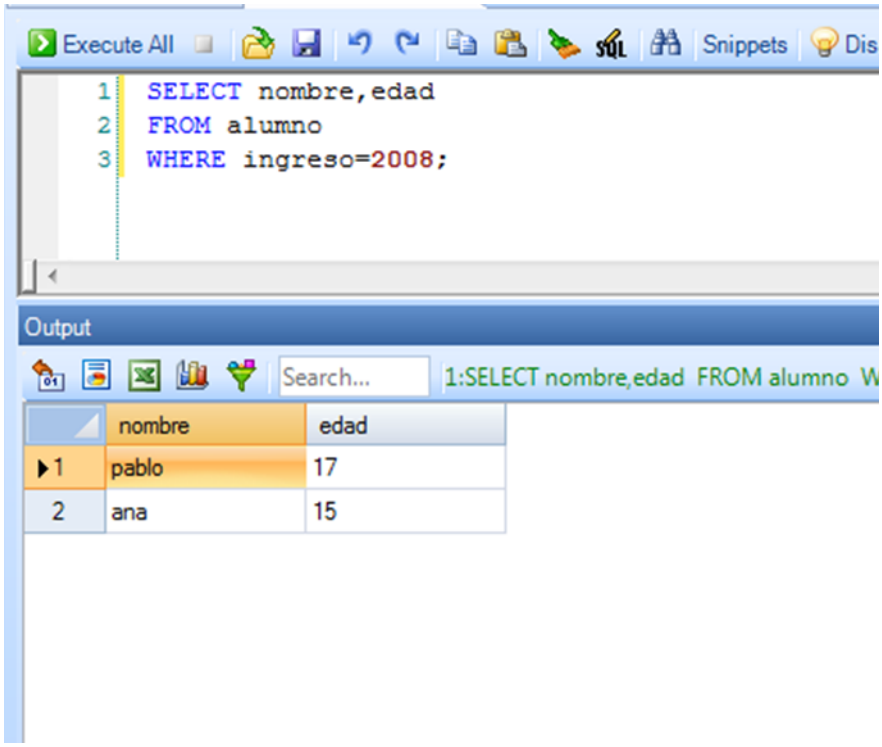
3.4 Traductores: Compiladores e Intérpretes

Los compiladores, los intérpretes y los ensambladores se encargan de traducir lo que haya escrito en lenguaje de alto nivel (código fuente) y lo convierten a código objeto (casi ejecutable).

Dado que la computadora solo ejecuta lenguaje de máquina, los programas escritos en lenguajes de alto nivel o ensamblador deben ser traducidos. Esta tarea la realizan

ISBN: 978-631-6960-03-0

Figura 3.5: Ejemplo de la Sentencia SQL



Fuente: (Gutiérrez, 2011)

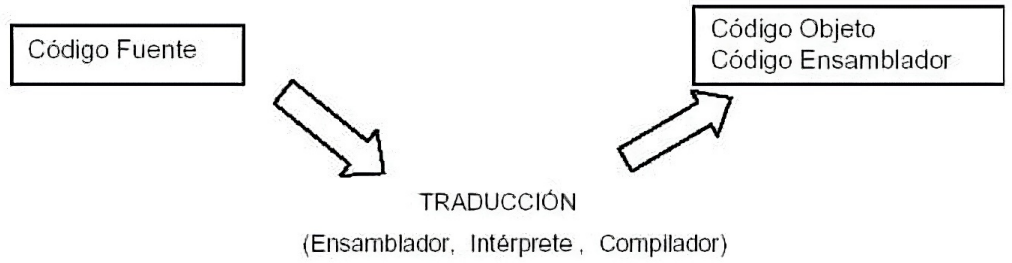
Figura 3.6: Ejemplo de la Prolog

```
/* Equivale a:
Para todo X e Y, si X es mujer y X es el progenitor de Y,
entonces X es la madre de Y */
madre(X,Y):-
    mujer(X),
    progenitor(X,Y).

/* Equivale a:
Para todo X e Y, si X es hombre y X es el progenitor de Y,
entonces X es el padre de Y */
padre(X,Y):-
    hombre(X),
    progenitor(X,Y).

%Hechos.
mujer(clara).
mujer(isabel).
mujer(ana).
mujer(patricia).
hombre(tomas).
hombre(jose).
hombre(jaime).
```

Fuente: (Pineda et al., 2025)

ISBN: 978-631-6960-03-0

Figura 3.7: Traducción de Código Fuente a Código Objeto



Elaboración propia.

programas especiales llamados traductores, principalmente compiladores e intérpretes.

3.4.1 Compiladores

Es un programa que traduce un programa escrito en un lenguaje de alto nivel, por ejemplo, C++, en un programa en lenguaje de máquina que la computadora es capaz de entender y ejecutar directamente. Un compilador es un tipo especial de programa, en cuanto a que sus entradas o datos son algún programa y su salida es otro programa. Para evitar confusiones, solemos llamar programa fuente o código fuente al programa de entrada, y programa objeto o código objeto a la versión traducida que el compilador produce. Código se usa frecuentemente para referirse a un programa o a una parte de él, sobre todo cuando se habla de programas objeto. Ejemplo

- Pascal
- Cobol
- Fortran
- Ada
- Código Fuente
- Código Objeto
- Código Ensamblador
- Modula 2
- C

ISBN: 978-631-6960-03-0

- C++

El compilador, informa al usuario de la presencia de errores en el programa fuente, pasándose a crear el programa objeto cuando está libre de errores. El código objeto puede ser ejecutado posteriormente. Una vez traducido un programa, su ejecución es independiente de su compilación. Involucra dos pasos en su operación:

1. **Convertir código fuente a objeto**
2. **Ejecutar el código objeto**

Ventaja: Al tener el código objeto, el programa se ejecuta más rápido

Fases de compilación

Análisis: Dependiente del lenguaje. Independiente de la máquina
Sintaxis: Independiente del lenguaje. Dependiente de la máquina.

3.4.2 Intérpretes

Es el que permite que un programa fuente escrito en un lenguaje vaya traduciéndose y ejecutándose directamente sentencia a sentencia por el computador. Convierte uno por uno los enunciados del código fuente a código objeto antes de ser ejecutados. Convierte y ejecuta el programa en línea al mismo tiempo.

Ejemplo: **Basic estándar.**

Ventajas

- Resulta más fácil localizar y corregir errores (depuración de programas)
- Son más pedagógicos para aprender a programar.
- El programa es más fácil de desarrollar

Traducen programas de alto nivel. No se genera en la mayoría de los ficheros.

- Programa
- Fuente

ISBN: 978-631-6960-03-0

- Código
- Intermedio
- Programa
- Objeto

Para cada una de las líneas se ejecuta el siguiente proceso:

a. *Análisis de la instrucción de esa línea*

b. *Traducción de esa línea (si ya está correcta) a código objeto*

c. *Ejecución de esa línea*

Con el intérprete, cada vez que necesitamos ejecutar el programa tenemos que volver a analizarlo porque no hay código objeto.

Con el compilador, aunque más lenta la traducción, sólo se realiza una vez.

Tabla 3.1: Comparación entre compiladores e intérpretes

Característica	Compilador	Intérprete
Proceso	Traducción completa previa a la ejecución.	Traducción y ejecución línea por línea.
Velocidad de Ejecución	Alta (el código máquina se ejecuta directamente).	Baja (debe traducir en tiempo de ejecución).
Detección de Errores	Reporta todos los errores de sintaxis antes de generar el ejecutable.	Se detiene al encontrar el primer error durante la ejecución.
Portabilidad del Ejecutable	Baja (depende de la plataforma para la que se compiló).	Alta (el mismo código fuente puede correr donde haya intérprete).

Elaboración propia.

Ejemplo:

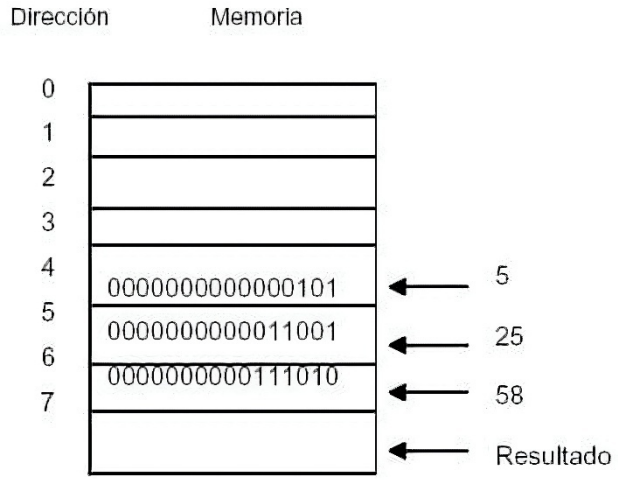
Supongamos que se han almacenado tres valores 5 = 01012, 25 = 110012 y 58 = 1110102 en las posiciones de memoria con direcciones 4, 5 y 6.

Queremos multiplicar los dos primeros valores, sumar el tercero y almacenar el resultado en la palabra de memoria 7.

Para llevar a cabo este cálculo, se deben ejecutar las siguientes instrucciones:

ISBN: 978-631-6960-03-0

Figura 3.8: Memoria de los archivos 5 - 25 – 58



Elaboración propia.

1. Recuperar el contenido de la palabra de memoria 4 y cargarlo en el registro acumulador de la unidad aritmético - lógica.

2. Recuperar el contenido de la palabra de memoria 5 y calcular el producto de este valor y el valor situado en el acumulador.

3. Recuperar el contenido de la palabra de memoria 6 y sumar su valor con el valor situado en el registro acumulador.

4. Almacenar el contenido del registro acumulador en la palabra de memoria 7. Para almacenar estas instrucciones en la memoria de la computadora, deben estar representadas en forma binaria. Las direcciones de los datos no presentan problemas, puesto que pueden ser convertidos fácilmente a direcciones binarias:

   4 = 100
5 = 101
6 = 110
7 = 111

5. Las operaciones de cargar, multiplicar, sumar, almacenar y otras instrucciones de máquina básicas se representan mediante códigos numéricos, llamados códigos de operación, por ejemplo:

ISBN: 978-631-6960-03-0

LOAD = 16 = 10000
STORE = 17 = 10001
ADD = 35 = 100011
MULTIPLY = 36 = 100100
SUB = 37 = 100101
DIV = 38 = 100110

6. Usando parte de una palabra para almacenar el código de operación y otra para la dirección del operando, podemos representar nuestra secuencia de instrucciones en lenguaje máquina como sigue:

1. 0001000000000100
2. 0010010000000101
3. 0010001100000110
4. 0001000100000111

Estas pueden ser almacenadas en cuatro palabras consecutivas de memoria. Cuando se ejecuta el programa, la unidad de control recuperará cada una de las instrucciones, la decodificará para determinar la operación y la dirección del operando, recuperará el operando, y entonces ejecutará la operación requerida, usando la unidad aritmético lógica cuando sea necesario. Los programas para las primeras computadoras tuvieron que ser escritos en lenguaje de máquina.

Posteriormente fue posible escribirlos en lenguaje ensamblador usando códigos nemotécnicos en lugar de códigos de operación numéricos y nombres de variables en lugar de direcciones numéricas.

Por ejemplo la secuencia de instrucciones anteriores se escribiría así:

1. LOAD A
2. MULT B
3. ADD C
4. STORE X

Luego que se crearon los lenguajes de alto nivel, las instrucciones se escribían en forma más entendible para el programador. El ejemplo anterior podría ser como lo siguiente usando C++:

X = A * B + C

El producto en la programación se representa por asterisco. En cada uno de estos casos (ensamblador y lenguajes de alto nivel), el compilador traduce cada instrucción

ISBN: 978-631-6960-03-0

del programa en una secuencia de cuatro instrucciones máquina y genera un programa objeto.

Ejercicios Propuestos:

Usando mnemónicos de instrucción y códigos de operación, escriba una secuencia de instrucciones en a) Lenguaje ensamblador b) Lenguaje de máquina, equivalente a las instrucciones de C++:

1. X = (A - B) * C

X = (A+B) / (C+D)

X = (A + B) - C

Para las instrucciones máquina, suponga que los valores de A, B, C y D son almacenados en las palabras de memoria 15,16, 17 y 18 respectivamente, y que los valores de X e Y se almacenarán en la palabra de memoria 23 y 24 respectivamente.

Este capítulo exploró la naturaleza y evolución de los lenguajes de programación. Se desglosaron sus componentes esenciales léxico, sintaxis y semántica que rigen su forma y significado. Se estableció la clasificación fundamental entre lenguajes de bajo nivel (cercanos a la máquina) y de alto nivel (cercanos al humano), y se explicó el rol crucial de los traductores, diferenciando entre compiladores e intérpretes. Finalmente, se recorrió la evolución histórica a través de las cinco generaciones de lenguajes, desde el lenguaje de máquina binario hasta los lenguajes orientados a problemas e inteligencia artificial. Esta comprensión permite apreciar las herramientas disponibles y seleccionar la más apropiada según los requerimientos de eficiencia, portabilidad y dominio del problema a resolver.

ISBN: 978-631-6960-03-0

CAPÍTULO 4

ALGORITMOS Y DIAGRAMAS DE FLUJO

4.1 Introducción a los Algoritmos

Un algoritmo es el corazón de la solución programada. Se define como una serie finita de pasos lógicos, ordenados y no ambiguos que, al seguirse, resuelven un problema específico (Cairó Battistutti, 2005). Es la receta detallada que convierte una especificación abstracta en acciones concretas ejecutables por una máquina.

4.1.1 Definición de Algoritmo

Es una serie de operaciones detalladas a ejecutar paso a paso, que conducen a la resolución de problemas. Es un conjunto de reglas para resolver determinado problema describiendo de forma lógica su solución. Cada una de las acciones de que consta un algoritmo es denominada sentencia y éstas deben ser escritas en términos de cierto lenguaje comprensible para el computador, que es el lenguaje de programación. Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en que han de ser ejecutadas.

4.1.2 Características de un Algoritmo

Todo algoritmo debe poseer las siguientes características (Joyanes Aguilar, 1996):

- **Precisión:** Cada paso debe estar definido de manera clara y sin ambigüedad.
- **Determinismo:** Dada la misma entrada, el algoritmo siempre produce la misma salida.
- **Finitud:** Debe terminar después de un número finito de pasos.
- **Entrada:** Puede tener cero o más datos de entrada definidos externamente.
- **Salida:** Debe producir al menos un resultado.

ISBN: 978-631-6960-03-0

- **Efectividad:** Cada operación debe ser lo suficientemente básica como para poder ejecutarse de manera exacta y en un tiempo finito.

4.1.3 Partes de un Algoritmo

Estructuralmente, un algoritmo se compone de tres partes fundamentales:

Tabla 4.1: Estructura básica de un algoritmo

Entrada	Proceso	Salida
Los datos que el algoritmo recibirá para procesar.	La secuencia lógica de operaciones (cálculos, comparaciones, asignaciones) que transforman la entrada.	Los resultados generados después del procesamiento.

Elaboración propia.

4.2 Herramientas de Representación Algorítmica

Antes de codificar en un lenguaje específico, es crucial representar el algoritmo de manera clara y comprensible. Existen diversas herramientas para este fin, cada una con sus ventajas.

4.2.1 Descripción Narrada

Este algoritmo es caracterizado porque sigue un proceso de ejecución común y lógico, describiendo textualmente paso a paso cada una de las actividades a realizar dentro de una actividad determinada.

Ejemplo: Algoritmo para asistir a clases:

1. Levantarse
2. Bañarse
3. Vestirse
4. Desayunar
5. Cepillarse los dientes

ISBN: 978-631-6960-03-0

6. Salir de casa
7. Tomar el autobús
8. Llegar a la ESPOCH
9. Buscar el aula
10. Ubicarse en un asiento

4.2.2 Pseudocódigo

Pseudo = falso.

El pseudo código no es realmente un código sino una imitación y una versión abreviada de instrucciones reales para las computadoras. Es una técnica para diseño de programas que permite definir las estructuras de datos, las operaciones que se aplicarán a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema. Utiliza un pseudolenguaje muy parecido a nuestro idioma, pero que respeta las directrices y los elementos de los lenguajes de programación. Se concibió para superar las dos principales desventajas de los flujogramas: lento de crear y difícil de modificar sin un nuevo redibujo.

Ejemplo:

Diseñar un algoritmo que lea cuatro variables y calcule e imprima su producto, suma y media aritmética.

Inicio
leer (a, b, c, d)
producto
textless– (a * b * c * d)
suma
textless– (a + b + c + d)
media
textless– (a + b + c + d) / 4
escribir (producto, suma, media)
Fin

ISBN: 978-631-6960-03-0

Figura 4.1: Ejemplo de Pseudocódigo en PSEINT

```
1  Algoritmo SumaDosNumeros
2      Definir num1, num2, resultado Como Real
3      Escribir "Ingrese el primer número:"
4      Leer num1
5      Escribir "Ingrese el segundo número:"
6      Leer num2
7      resultado <- num1 + num2
8      Escribir "La suma es: ", resultado
9  FinAlgoritmo
```

Elaboración propia.

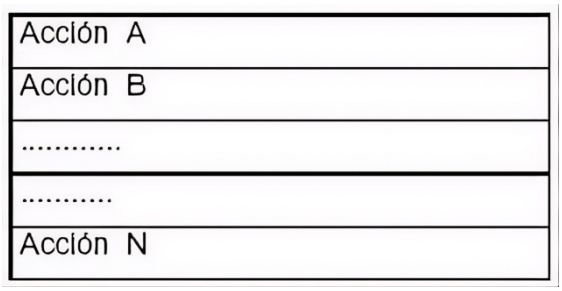
ISBN: 978-631-6960-03-0

4.2.3 Diagramas N-S (Nassi-Schneiderman)

Son una herramienta que favorece la programación estructurada y reúne características gráficas propias de diagramas de flujo y lingüísticas propias de pseudocódigos. Constan de una serie de cajas contiguas que se leerán siempre de arriba-abajo y sus estructuras lógicas son las siguientes:

Estructura Secuencial

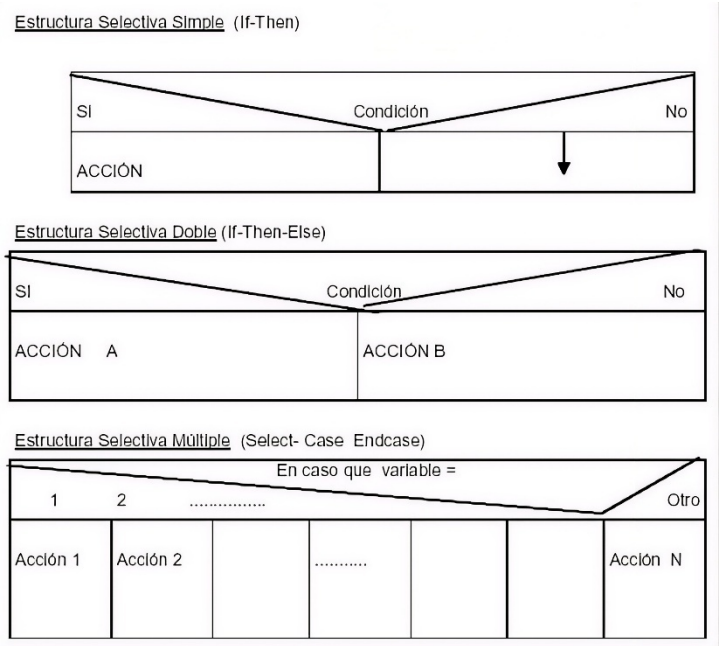
Figura 4.2: Estructura Secuencial de Diagrama N-S



Elaboración propia.

Estructura Condicional

Figura 4.3: Estructura Condicional del Diagrama N-S

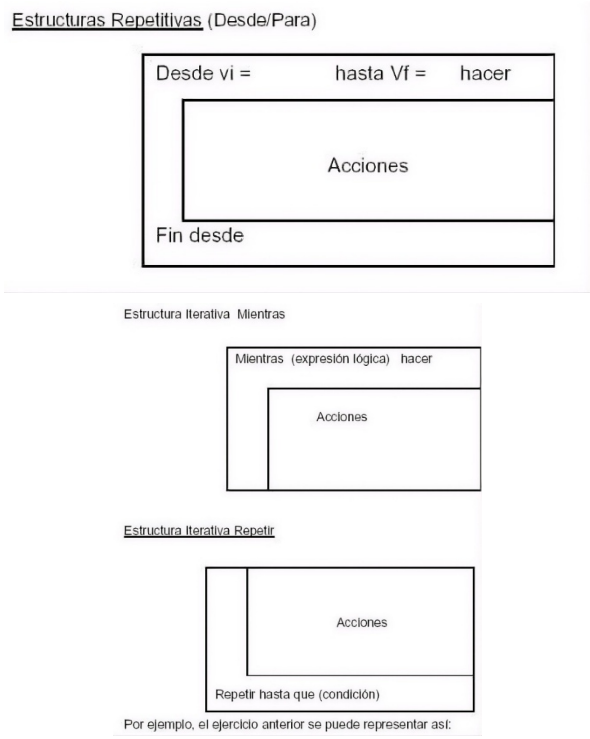


Elaboración propia.

ISBN: 978-631-6960-03-0

Estructura Repetitiva

Figura 4.4: Estructura Repetitiva del Diagrama N-S



Elaboración propia.

Siguiendo el ejemplo anterior se puede representar de esta manera

4.2.4 Diagramas de Flujo

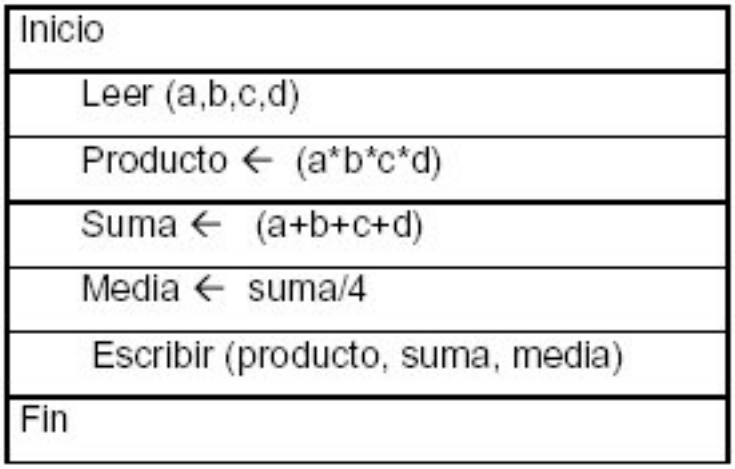
Son la representación gráfica de la solución algorítmica de un problema. Para diseñarlos se utilizan determinados símbolos o figuras que representan una acción dentro del procedimiento. Utilizan unos símbolos normalizados, con los pasos del algoritmo escritos en el símbolo adecuado y los símbolos unidos con flechas, denominadas líneas de flujo, que indican el orden en que los pasos deben ser ejecutados.

Para su elaboración se siguen ciertas reglas:

- Se escribe de arriba hacia abajo y de izquierda a derecha
- Siempre se usan flechas verticales u horizontales, jamás curvas
- Evitar cruce de flujos

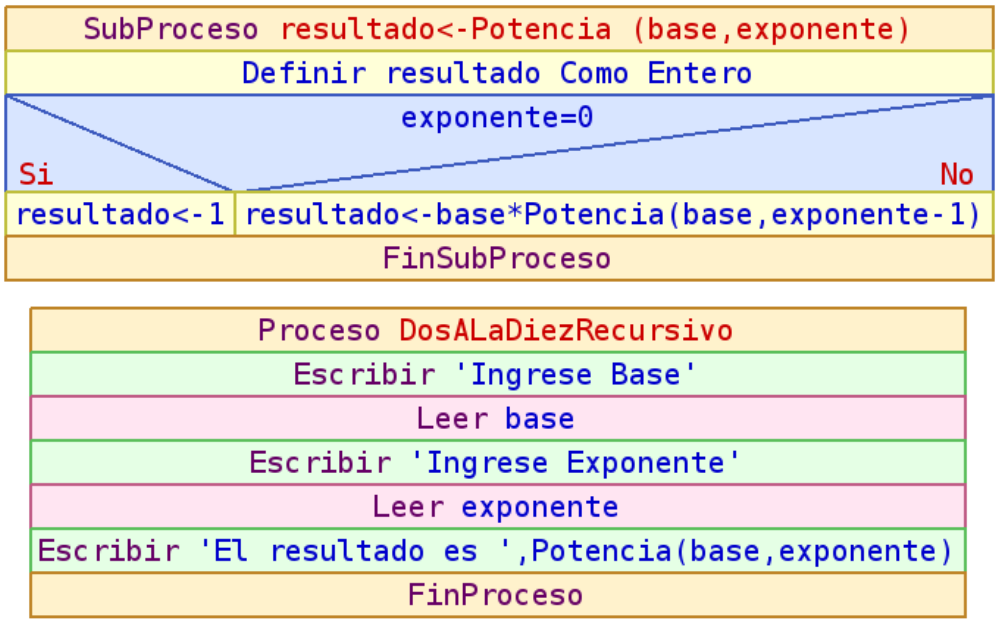
ISBN: 978-631-6960-03-0

Figura 4.5: Ejemplo de Operaciones básicas con el diagrama N-S



Elaboración propia.

Figura 4.6: Ejemplo de Pseint del diagrama N-S



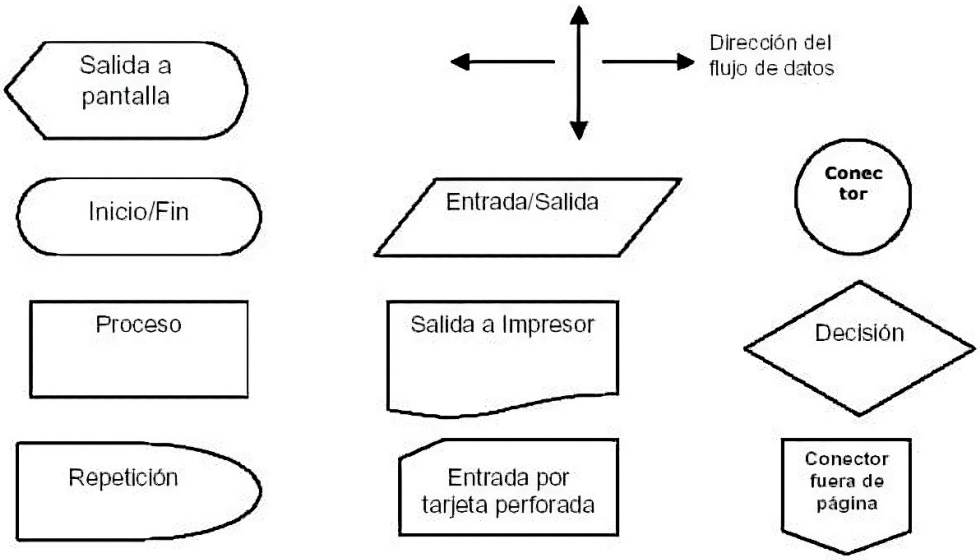
Fuente: (pseint, 2025)

ISBN: 978-631-6960-03-0

- En cada paso expresar una acción concreta
- Secuencia de flujo normal en una solución de problema
- Tiene un inicio
- Una lectura o entrada de datos
- El proceso de datos
- Una salida de información
- Un final

Simbología para diseñar flujogramas

Figura 4.7: Simbología para el diseño de flujogramas



Elaboración propia.

Ventajas de usar Flujogramas

- Rápida comprensión de las relaciones
- Análisis efectivo de las diferentes secciones del programa

ISBN: 978-631-6960-03-0

- Pueden usarse como modelos de trabajo en el diseño de nuevos programas o sistemas
- Comunicación con el usuario
- Documentación adecuada de los programas
- Codificación eficaz de los programas
- Depuración y pruebas ordenadas de programas

Desventajas de usar Flujogramas

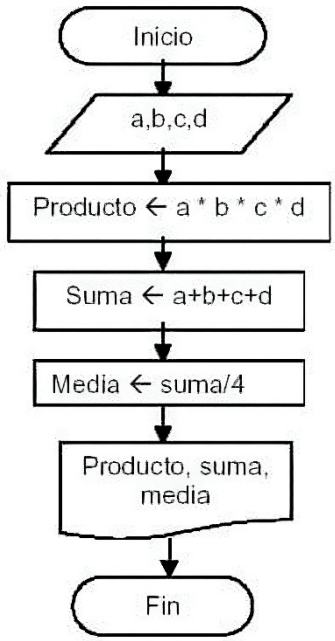
- Diagramas complejos y detallados suelen ser laboriosos en su planteamiento y diseño
- Acciones a seguir tras la salida de un símbolo de decisión, pueden ser difíciles de seguir si existen diferentes caminos
- No existen normas fijas para la elaboración de los diagramas de flujo que permitan incluir todos los detalles que el usuario desee introducir.

Representando el ejemplo como flujograma tenemos:

Este capítulo estableció el vínculo esencial entre la lógica formal y la programación. Se definieron los fundamentos del razonamiento lógico, incluyendo argumentos, premisas y conclusiones, y se demostró su materialización práctica en las expresiones booleanas y tablas de verdad, pilares de la toma de decisiones en el código. Posteriormente, se introdujo el concepto central de algoritmo, detallando sus características y partes constitutivas. Finalmente, se presentaron las principales herramientas para representar algoritmos como lo es la descripción narrada, pseudocódigo, diagramas de flujo y diagramas N-S, cada una adecuada para distintas etapas del diseño.

ISBN: 978-631-6960-03-0

Figura 4.8: Ejemplo del Flujograma



Elaboración propia.

ISBN: 978-631-6960-03-0

CAPÍTULO 5

TIPOS DE DATOS, VARIABLES Y OPERACIONES

En programación, los datos son la materia prima y las operaciones son las herramientas que los transforman. Este capítulo introduce los elementos fundamentales que permiten manipular información dentro de un programa: los tipos de datos que definen la naturaleza de la información, las variables y constantes que la almacenan, y los operadores que permiten realizar cálculos y comparaciones. Comprender estos conceptos es esencial para escribir programas efectivos y libres de errores.

5.1 Dato

Es la expresión general que describe los objetos con los cuales opera el programa. Por ejemplo, la edad y el domicilio de una persona forman parte de sus datos. Los datos se sitúan en objetos llamados variables.

5.2 Tipo

Es el conjunto de valores que puede tomar una variable. Es el conjunto de transformaciones y funciones internas y externas definidas sobre el conjunto de datos. Se tienen dos tipos de datos: Simples como numéricos y alfanuméricos y Estructuras de datos que pueden ser internas o externas.

5.3 Tipos de Datos Primitivos

Los tipos de datos primitivos o básicos son los tipos de datos predefinidos en un lenguaje de programación, que representan valores simples e indivisibles. Son los bloques de construcción para formar estructuras de datos más complejas.

5.3.1 Numéricos

Son aquellos cuyo contenido es una serie de dígitos (0-9) que en conjunto nos proporcionan un valor numérico ya sea entero o real y pueden ser precedidos de un signo

ISBN: 978-631-6960-03-0

+ ó -. Se subdivide principalmente en:

- **Enteros (int):** Representan números sin parte fraccionaria, positivos o negativos, donde ocupan un espacio fijo en memoria.
- **Reales (float, double):** Representan números con parte fraccionaria donde se utiliza notación de coma flotante, lo que implica una posible aproximación en valores.

Tabla 5.1: Tipos de datos numéricos comunes

Tipo	Ejemplo de Declaración	Valores Ejemplo	Uso Típico
Entero	int edad;	25, -10, 0	Contadores, índices, años.
Real (precisión simple)	float promedio;	85.5, 3.14, -2.1	Medidas, cálculos científicos.
Real (precisión doble)	double pi;	3.1415926535	Cálculos de alta precisión.

Elaboración propia.

5.3.2 Alfanuméricos

Son aquellos cuyo contenido son letras del abecedario, números o caracteres especiales o bien una combinación de ellos, donde se subdividen en:

- **Carácter (char):** Almacena un solo símbolo (letra, dígito, signo de puntuación) según un código como ASCII o Unicode (ej., 'A', '7', '$').
- **Cadena (string):** Representa una secuencia ordenada de caracteres (ej., "Hola Mundo", Çalle 123"). Es una estructura compuesta de caracteres primitivos.

5.3.3 Lógicos (Booleanos)

Este tipo, llamado bool o booleano, representa solo dos valores de verdad: **Verdadero (true, 1)** o **Falso (false, 0).** Es el resultado fundamental de las operaciones de comparación y lógica, y controla el flujo de ejecución en estructuras condicionales y bucles.

ISBN: 978-631-6960-03-0

5.4 Variables, Constantes e Identificadores

Para trabajar con datos, necesitamos referenciarlos en la memoria de la computadora mediante nombres simbólicos.

5.4.1 Variables

Son zonas de memoria cuyo contenido cambia durante la fase de procesamiento de información. Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa. Las variables llevan un nombre llamado identificador. Este puede ser una cadena de letras y dígitos, empezando siempre con una letra; Por ejemplo:

Pi, curso99, nom_alum, etc.

5.4.2 Constantes

Una constante es similar a una variable, pero su valor se define una vez y no puede modificarse después. Se utilizan para representar valores fijos y significativos en el programa, mejorando la claridad y mantenibilidad. Ejemplo:

const float IVA = 0.19;

5.4.3 Identificadores

Son palabras creadas por los programadores para dar nombre a los objetos y demás elementos que necesitamos declarar en un programa: variables, constantes, tipos, estructuras de datos, archivos, subprogramas, etc. En C++ las letras mayúsculas se tratan como diferentes y distintas unas de otras. Por ejemplo, contador, Contador y CONTADOR son tres nombres de identificadores distintos. Un identificador no puede ser igual a una palabra reservada, y no debe tener el mismo nombre que una función, ya sea definida por el usuario o de la biblioteca de C. Constantes: Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo.

Por ejemplo, si se desea obtener un reporte para cada uno de los empleados de una empresa, con sus datos generales, la fecha y cantidad de dinero que recibieron la última semana, el dato fecha puede ser una constante ya que es el mismo para todos.

ISBN: 978-631-6960-03-0

5.5 Tipos de Instrucciones

a. *Instrucciones de inicio/fin*

(ejemplo inicio - fin)

b. *Instrucciones de asignación*

(ejemplo B ! 7)

c. *Instrucciones de lectura (entrada)*

(ejemplo leer)

d. *Instrucciones de escritura (salir)*

(ejemplo escribir)

e. *Instrucciones de bifurcación*

(ejemplo Goto fin)

5.6 Expresiones y Operadores

Son representaciones de un cálculo necesario para la obtención de un resultado. Estas pueden ser valores constantes, funciones o combinaciones de valores, variables, funciones y operadores que cumplen determinadas reglas de construcción de una expresión. Son un conjunto de operadores y operandos que producen un valor; Por ejemplo:

Cos (pi * X) + 12.56 * SQR(100)

Un operador es un símbolo o palabra que significa que se ha de realizar cierta acción entre uno o dos valores que son llamados operandos.

ISBN: 978-631-6960-03-0

5.7 Tipos de Operadores

a. **Aritméticos (su resultado es un número):**

   potencia, * , / , mod, div, + , -

b. **Relacionales (su resultado es un valor de verdad):**

   =, <, >, <=, >=, <>

c. **Lógicos o Booleanos (su resultado es un valor de verdad):**

   not, and, or

d. **Alfanuméricos :**

   + (concatenación)

e. **Asociativos.** Es el paréntesis ( ), permite indicar en qué orden deben realizarse las operaciones. Cuando una expresión se encuentra entre paréntesis, indica que las operaciones que están dentro de ellos deben realizarse primero. Si en una expresión se utilizan más de un paréntesis se deberá proceder primero con los que se encuentren más hacia el centro de la expresión.

5.7.1 Operadores Aritméticos

Realizan cálculos matemáticos básicos con operandos numéricos. Producen un resultado numérico.

Suma (+), Resta (-), Multiplicación (*), División (/).

Módulo (%): Devuelve el residuo de una división entera.

5.7.2 Operadores Relacionales

Comparan dos valores y devuelven un resultado booleano (true o false). Son fundamentales para las decisiones.

Igual a (==), Diferente de (!=).

Mayor que (>), Menor que (<), Mayor o igual (>=), Menor o igual (<=).

ISBN: 978-631-6960-03-0

5.7.3 Operadores Lógicos

Operan sobre valores booleanos (o expresiones que los producen) para formar condiciones compuestas.

- **Y lógico (&&, AND):** true solo si ambos operandos son true.
- **lógico (||, OR):** true si al menos un operando es true.
- **Negación (!, NOT):** Invierte el valor booleano (!true es false).

5.7.4 Operadores de Asignación

Asignan un valor a una variable. El más básico es el simple (=). Existen versiones compuestas que combinan una operación con la asignación. Por ejemplo:

total += precio; equivale a total = total + precio;

5.8 Jerarquía de Operadores

Cuando una expresión contiene varios operadores, la jerarquía o precedencia de operadores determina el orden en que se ejecutan. Evaluar una expresión sin conocer estas reglas conduce a resultados incorrectos.

5.8.1 Reglas de precedencia

El orden general de precedencia (de mayor a menor) es:

- Paréntesis () (siempre tienen la máxima prioridad).
- Operadores unarios (como negación !, signo negativo -).
- Operadores aritméticos: Multiplicación/División/Módulo (*, /, %) antes que Suma/Resta (+, -).
- Operadores relacionales (>, <, >=, <=).
- Operadores de igualdad (==, !=).
- Operadores lógicos: AND (&&) antes que OR (||).

ISBN: 978-631-6960-03-0

- Operadores de asignación (=, +=, etc.).

Datos de tipo entero tienen los operadores +, -, *, /, div, mod, abs, sqr, sqrt, ln, exp, sin, cos, tan, pow, etc. Los datos de tipo real tienen los mismos operadores enteros y además trunc, round, int, y otros. La suma y multiplicación de datos de tipo real cumplen la propiedad conmutativa, pero no siempre la asociativa y la distributiva.

Para resolver una expresión aritmética se deben seguir las siguientes reglas:

- Primero se resuelven las expresiones que se encuentran entre paréntesis.
- Se procede aplicando la jerarquía de operadores.
- Al evaluar una expresión, si hay dos operadores con la misma jerarquía, se procede a evaluar de izquierda a derecha.
- Si hay expresiones relacionales, se resuelven primero paréntesis, luego se encuentran los valores de verdad de las expresiones relacionales y por último se aplica jerarquía de operadores lógicos. En caso de haber iguales, proceder de izquierda a derecha.

5.8.2 Uso de paréntesis

Los paréntesis () permiten anular la precedencia natural y explicitar el orden de evaluación deseado, mejorando la legibilidad y evitando ambigüedades. Es una buena práctica usar paréntesis incluso cuando no sean estrictamente necesarios, si la expresión podría no ser clara.

EJERCICIOS

Aplicando la jerarquía de los operadores, encontrar el valor de cada una de las siguientes expresiones:

1) 4 + 1 * 5 ^ 2 - 1
2) 9 / 3 + 4 ^ 2 - 5 * 1 + 9 / -2 + 3
3) 5 / 2 + 3 - 4 * 5 / 2
4) (4 + 1) * 5 ^ 2 - 1
5) 17 / 2 + 3 ^ 2 ^ 2 - 2 * 2 / 2

Aplicando la jerarquía de operadores, encontrar el valor de verdad de cada una de las siguientes expresiones:

ISBN: 978-631-6960-03-0

1. Not ((M > N and R > S) or (NOT(T < V and S > M))) para M=8, N=9, R=5, S=5 , T=4 y V= 2

2. (3 * 2 ^ 2 - 4 / 2 * 1) > (3 * 2 ^ -4 / 2 * 1) and (5 > 9 / 3)

ISBN: 978-631-6960-03-0

CAPÍTULO 6

HERRAMIENTAS DE PROGRAMACIÓN

El presente capítulo aborda el estudio de diversas herramientas de programación que apoyan el proceso de enseñanza aprendizaje en los niveles iniciales, con el propósito de facilitar la comprensión de los fundamentos de la programación y el desarrollo del pensamiento lógico. En particular, se analizan herramientas didácticas y entornos de desarrollo que permiten a los estudiantes pasar progresivamente del diseño algorítmico a la implementación de soluciones computacionales, fortaleciendo así sus competencias en la resolución de problemas y en la construcción de programas (Beúnes Cañete et al., 2019).

6.1 Uso de las herramientas

La enseñanza de la programación en los niveles iniciales representa un desafío, puesto que los estudiantes deben asimilar simultáneamente conceptos lógicos, matemáticos y sintácticos. En este contexto, el uso de herramientas informáticas educativas fortalece el proceso de enseñanza aprendizaje, ya que proporciona entornos que facilitan la comprensión progresiva de los fundamentos de la programación.

6.1.1 PSeInt como herramienta para el desarrollo del pensamiento algorítmico

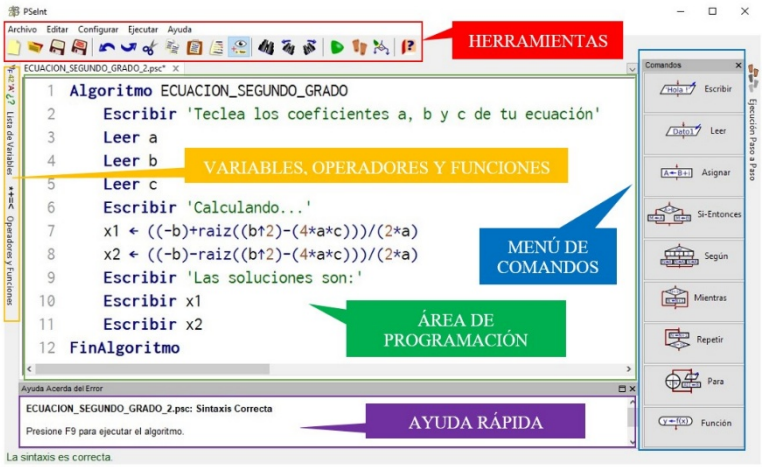
En primer lugar, PSeInt es una herramienta didáctica orientada al aprendizaje del pseudocódigo y la construcción de algoritmos. En otras palabras, permite que el estudiante se concentre principalmente en la lógica de resolución de problemas, sin enfrentar de manera inmediata la complejidad sintáctica de los lenguajes de programación.

Además, mediante el uso de PSeInt, los estudiantes desarrollan el pensamiento algorítmico, adquiriendo habilidades para analizar problemas, descomponerlos en partes y definir secuencias ordenadas de instrucciones. De igual modo, la herramienta facilita la comprensión de estructuras secuenciales, selectivas y repetitivas, las cuales constituyen la base de la programación.

Características

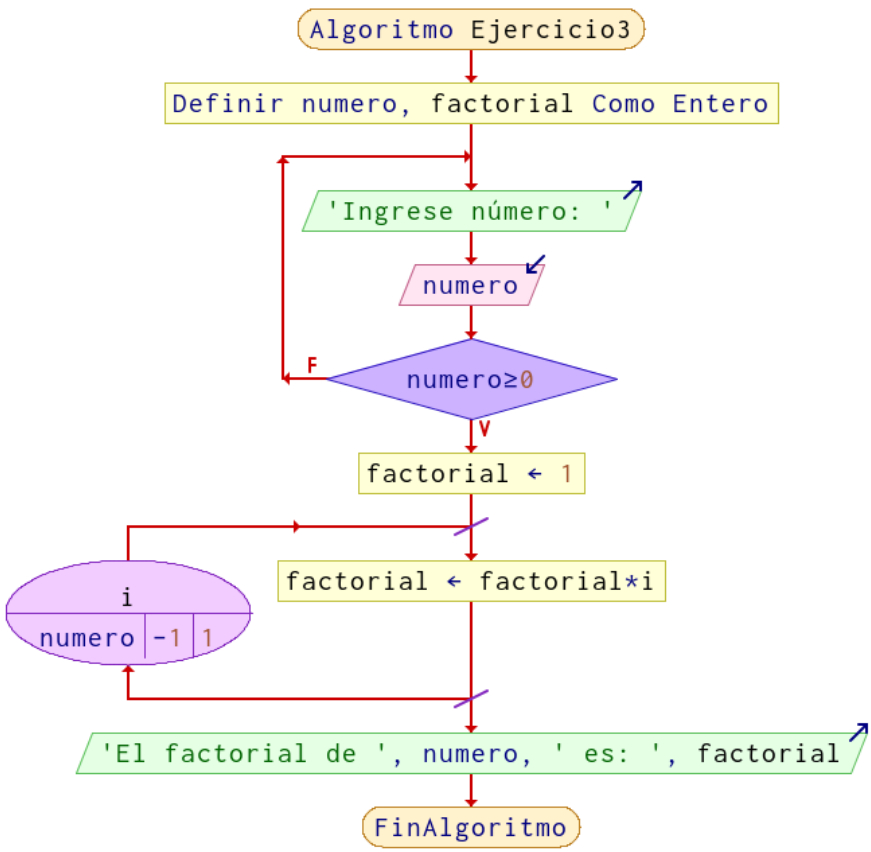
ISBN: 978-631-6960-03-0

Figura 6.1: Interfaz Gráfica de Pseint



Fuente: (INTEF, 2025)

Figura 6.2: Diagrama de Flujo creado con Pseint



Fuente: (Pseintlab, 2024)

ISBN: 978-631-6960-03-0

- Autocompletado
- Ayudas Emergentes
- Plantillas de Comandos
- Coloreado de Sintaxis
- Resaltado de bloques lógicos
- Indentado Inteligente
- Listados de funciones, operadores y variables

6.2 Visual Studio como entorno para la iniciación en la codificación

Por otra parte, Visual Studio es un entorno de desarrollo integrado (IDE) ampliamente utilizado para la creación de aplicaciones en diversos lenguajes de programación. Su incorporación en los cursos de Introducción a la Programación permite que los estudiantes se familiaricen con un entorno profesional de desarrollo.

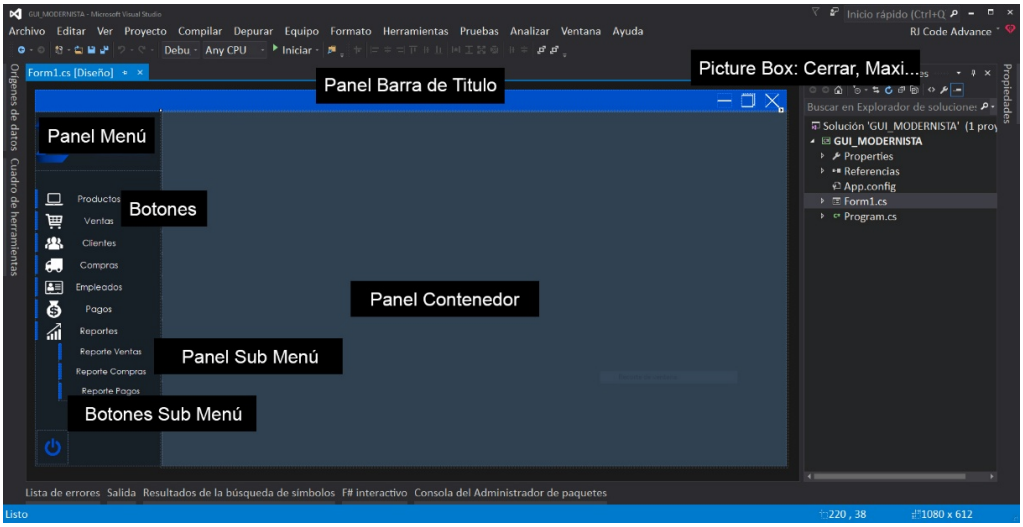
Así mismo, a través de Visual Studio, los estudiantes comienzan a comprender la estructura básica de un programa, la escritura de código fuente y la ejecución de aplicaciones. De esta manera, se fortalece la relación entre los algoritmos diseñados y su implementación práctica.

Características

- Interfaz intuitiva y altamente personalizable
- Soporte integrado para múltiples lenguajes de programación
- Sistema de extensiones para ampliar funcionalidades
- Control de versiones con integración Git nativa
- Depuración eficiente con herramientas integradas

ISBN: 978-631-6960-03-0

Figura 6.3: Interfaz Gráfica de Visual Studio



Fuente: (Advance, 2025)

6.3 Integración de PSeInt y Visual Studio en el proceso de aprendizaje

En segundo lugar, la combinación de PSeInt y Visual Studio establece una transición gradual entre el diseño algorítmico y la codificación. En una primera etapa, el estudiante formula la solución utilizando pseudocódigo en PSeInt; posteriormente, traduce dicha solución a un lenguaje de programación en Visual Studio.

Como resultado, este proceso favorece la consolidación de los conceptos de variables, tipos de datos, expresiones y estructuras de control, permitiendo que el estudiante comprenda cómo una idea lógica se transforma en un programa funcional.

Tabla 6.1: Comparativa de Pseint y Visual Studio

Criterio	PSeInt	Visual Studio
Tipo de herramienta	Software educativo para pseudocódigo	Entorno de Desarrollo Integrado (IDE)
Propósito principal	Diseñar y comprender algoritmos	Desarrollar programas en lenguajes reales
Nivel de dificultad	Bajo	Medio
Enfoque de aprendizaje	Pensamiento algorítmico y lógica	Codificación e implementación
Lenguaje utilizado	Pseudocódigo	C#, Visual Basic, C++, Python, entre otros
Orientación	Didáctica	Profesional
Tipo de entorno	Sencillo e intuitivo	Completo y configurable
Uso principal en el curso	Diseño de algoritmos	Traducción de algoritmos a código
Ventaja principal	Facilita la comprensión inicial	Permite crear aplicaciones reales

ISBN: 978-631-6960-03-0

Criterio	PSeInt	Visual Studio
Resultado en el estudiante	Comprende la lógica del problema	Implementa soluciones funcionales

Elaboración propia.

En primer lugar, la comparación entre PSeInt y Visual Studio permite evidenciar que ambas herramientas cumplen funciones complementarias en la enseñanza de la Introducción a la Programación. PSeInt se orienta al desarrollo del pensamiento algorítmico, ya que permite al estudiante comprender la lógica de los problemas mediante pseudocódigo. Por otra parte, Visual Studio facilita la implementación de los algoritmos en un lenguaje de programación real, fortaleciendo la comprensión de la estructura de un programa y la escritura de código; En conclusión, la integración de ambas herramientas favorece un aprendizaje progresivo, por consiguiente, contribuye a una formación más sólida en los fundamentos de la programación.

ISBN: 978-631-6960-03-0

CAPÍTULO 7

ESTRUCTURAS DE CONTROL DE FLUJO

El flujo de ejecución de un programa rara vez es una simple lista de instrucciones ejecutadas una tras otra. Para resolver problemas complejos, es necesario alterar este flujo secuencial basándose en condiciones o repitiendo bloques de código. Las estructuras de control son los elementos del lenguaje que permiten esta manipulación, dirigiendo el orden en que se ejecutan las instrucciones. Este capítulo presenta las tres categorías fundamentales de control de flujo: secuencial, selectiva y repetitiva, que constituyen la base de la lógica de cualquier programa.

Un problema se puede dividir en acciones elementales o instrucciones, usando un número limitado de estructuras de control (básicas) y sus combinaciones que pueden servir para resolver dicho problema.

Las Estructuras Básicas pueden ser:

- **Secuenciales:** Cuando una instrucción del programa sigue a otra.
- **Selección o decisión:** Acciones en las que la ejecución de alguna dependerá de que se cumplan una o varias condiciones.
- **Repetición, Iteración:** cuando un proceso se repite en tanto cierta condición sea establecida para finalizar ese proceso.

7.1 Estructuras Secuenciales

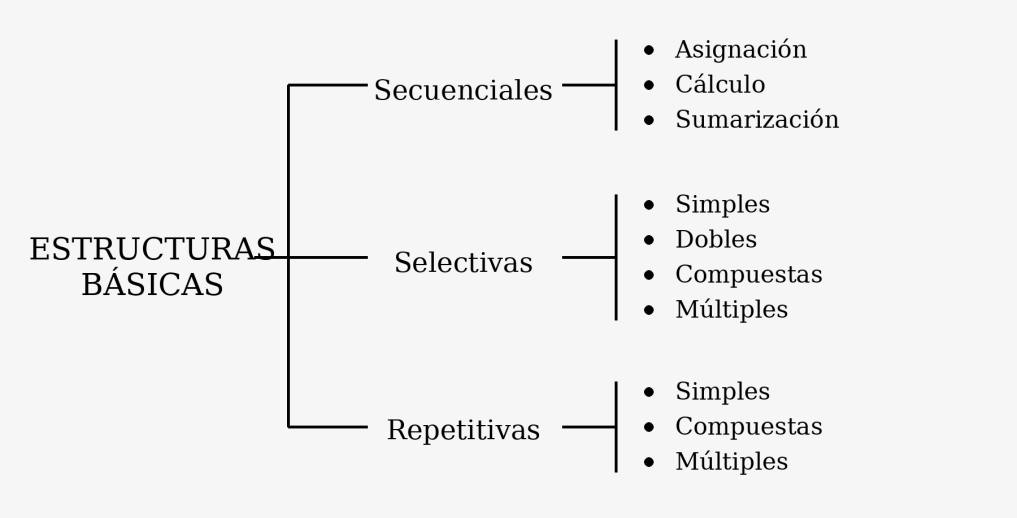
La estructura más simple y común es la secuencial, que sirve de base para las demás.

7.1.1 Definición

Se caracteriza porque una acción se ejecuta detrás de otra. El flujo del programa coincide con el orden físico en el que se han ido poniendo las instrucciones. Dentro de este tipo podemos encontrar operaciones de inicio/fin, inicialización de variables,

ISBN: 978-631-6960-03-0

Figura 7.1: Estructuras Básicas en Programación



Elaboración propia.

operaciones de asignación, cálculo, sumarización, etc. Este tipo de estructura se basa en las 5 fases de que consta todo algoritmo o programa

- Definición de variables (Declaración)
- Inicialización de variables.
- Lectura de datos
- Cálculo
- Salida

Ejemplo:

Un ejemplo clásico es un programa que lee dos valores, realiza una operación con ellos y muestra el resultado. Cada paso depende del anterior.

Otro ejemplo:

Se desea encontrar la longitud y el área de un círculo de radio 5.

Solución:

Representación en Diagrama de Flujo para el ejemplo:

Representación en Diagrama Nassi Schneiderman:

Los problemas secuenciales en diagramas N-S se representan solamente por cajas con líneas horizontales

ISBN: 978-631-6960-03-0

Figura 7.2: Secuencia Simple de un algoritmo

```
Algoritmo SecuenciaSimple
    Definir a, b, suma Como Entero
    Escribir "Ingrese el primer número:"
    Leer a
    Escribir "Ingrese el segundo número:"
    Leer b
    suma ← a + b // Esta línea espera que a y b ya tengan valor
    Escribir "La suma es: ", suma
FinAlgoritmo
```

Elaboración propia.

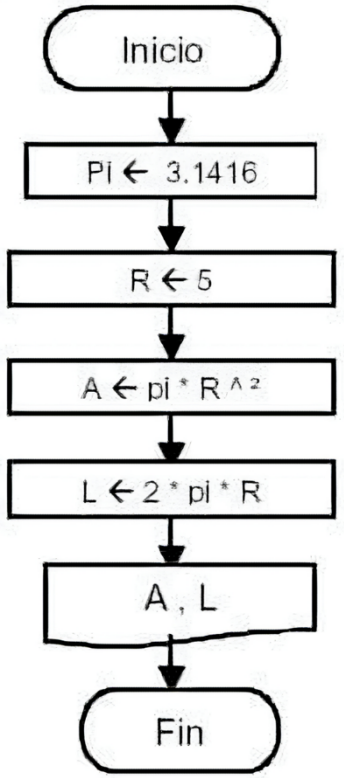
Figura 7.3: Ejemplo de Longitud de secuencia simple

```
Inicio
    Pi <- 3.1416 (definición de un valor constante)
    R <-  5 (radio constante ya que es conocido su valor)
    A <- pi * R ^ ² (asignación del valor del área)
    L <- 2 * pi * R (asignación del valor de la longitud)
    Escribir (A, L) (salida del algoritmo)
Fin
```

Elaboración propia.

ISBN: 978-631-6960-03-0

Figura 7.4: Ejemplo de Longitud de secuencia simple - Diagrama de Flujo



Elaboración propia.

Figura 7.5: Ejemplo de Longitud de secuencia simple - Diagrama N-S

Inicio
Pi ← 3.1416
R ← 5
A ← Pi * R ^ 2
L ← 2 * pi * R
Escribir (A, L)
Fin

Elaboración propia.

ISBN: 978-631-6960-03-0

7.2 Estructuras Selectivas

También llamadas estructuras de decisión o condicionales, permiten que el programa elija entre diferentes caminos de ejecución basándose en el valor de verdad (verdadero o falso) de una expresión lógica.

7.2.1 Estructura Simple (IF)

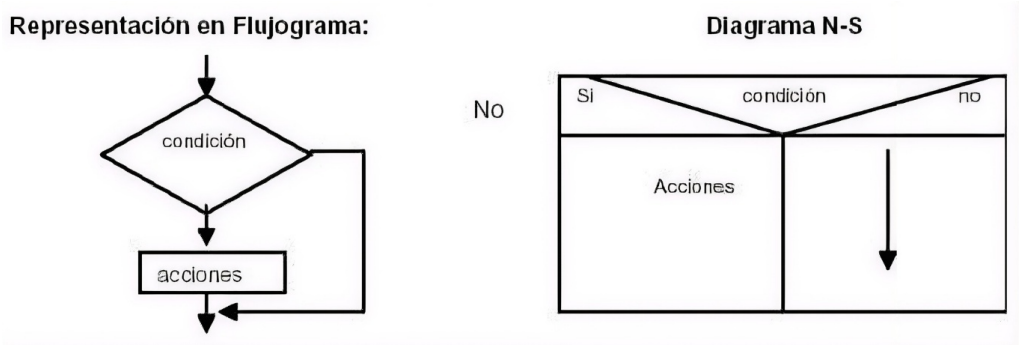
La estructura **SI (o IF)** ejecuta un bloque de instrucciones solo si una condición es verdadera. Si la condición es falsa, el bloque se omite y el programa continúa con la siguiente instrucción después de la estructura.

Figura 7.6: Estructura Selectiva Simple IF

```
1  SI (edad ≥ 18) ENTONCES
2      Escribir "Usted es mayor de edad."
3  FINSI
```

Elaboración propia.

Figura 7.7: Estructura Selectiva en representación de los Diagramas



Elaboración propia.

Ejemplo:

Construir un algoritmo tal, que dado como dato la calificación de un alumno en un examen, escriba .ªprobadoçuando la calificación es mayor que 8.

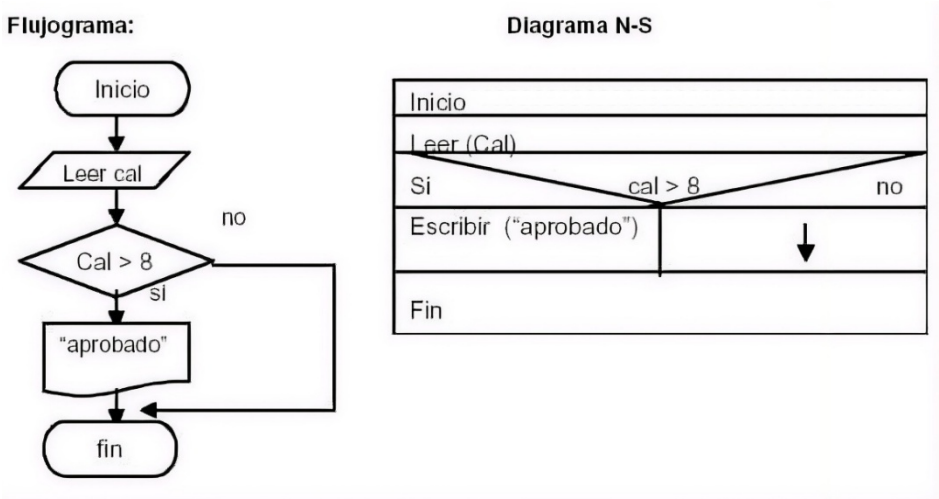
- Salidas: mensaje de aprobado si se cumple la condición.
- Entradas: calificación

ISBN: 978-631-6960-03-0

- Datos adicionales: un alumno aprueba si la calificación es mayor que 8
- Variables:
- Cal = calificación

Algoritmo
Inicio
Leer (cal)
Si cal > 8 entonces
Escribir (.ªprobado")
Fin_si
Fin
Diagramas

Figura 7.8: Diagrama N-S y de Flujo del Ejemplo para una Estructura Selectiva



Elaboración propia.

7.2.2 Estructura Doble (IF-ELSE)

La estructura **SI-SINO (o IF-ELSE)** permite elegir entre dos alternativas mutuamente excluyentes. Si la condición es verdadera, se ejecuta el bloque del **SI**; si es falsa, se ejecuta el bloque del **SINO**.

Diagrama de flujo con el Diagrama N-S

Ejemplo:

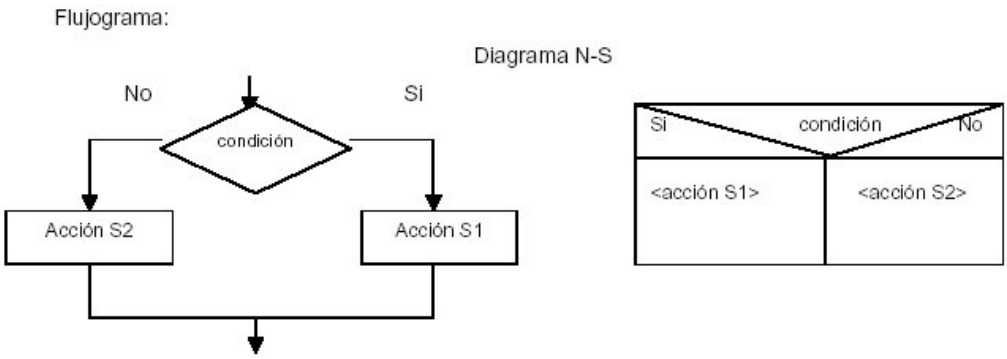
ISBN: 978-631-6960-03-0

Figura 7.9: Estructura Selectiva Doble IF-ELSE

```
1  SI (nota ≥ 60) ENTONCES
2      Escribir "Aprobado."
3  SINO
4      Escribir "Reprobado."
5  FINSI
```

Elaboración propia.

Figura 7.10: Estructura Selectiva en el Diagrama N-S



Elaboración propia.

ISBN: 978-631-6960-03-0

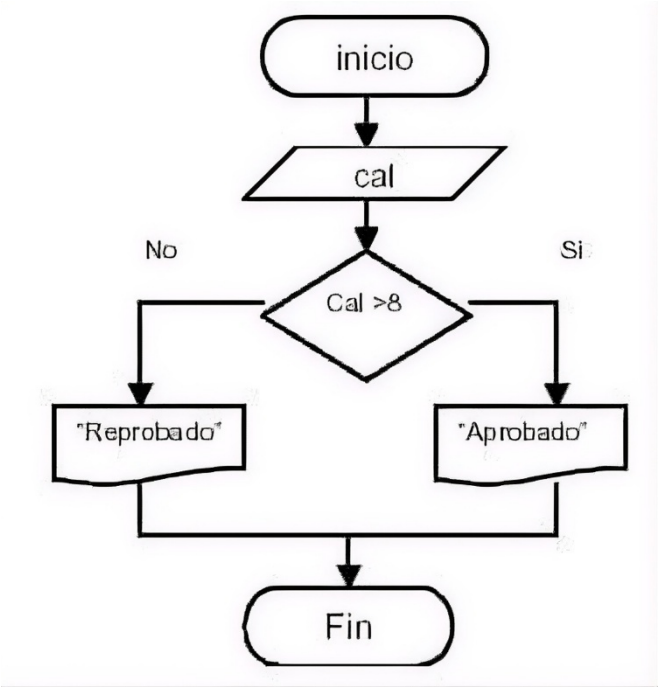
Dado como dato la calificación de un alumno en un examen, escriba .ªprobado"si su calificación es mayor que 8 y Reprobado.en caso contrario.

Algoritmo:

Inicio
Leer (cal)
Si cal > 8 entonces
Escribir (.ªprobado")
Sino
Escribir (reprobado")
Fin_si
Fin

Diagrama:

Figura 7.11: Estructura Selectiva - Ejemplo de aprobado



Elaboración propia.

ISBN: 978-631-6960-03-0

7.2.3 Estructuras Anidadas

Ocurre cuando una estructura selectiva **(SI o SI-SINO)** se coloca dentro del bloque de otra. Esto permite tomar decisiones más complejas y jerárquicas. Es fundamental una correcta indentación para mantener la legibilidad.

Figura 7.12: Estructura Selectiva Anidadas

```
SI (saldo > 0) ENTONCES
    SI (saldo ≥ monto_retiro) ENTONCES
        Escribir "Retiro exitoso."
    SINO
        Escribir "Fondos insuficientes."
    FINSI
SINO
    Escribir "Cuenta inactiva o sin fondos."
FINSI
```

Elaboración propia.

Ejemplo:

Dados los datos A, B y C que representan números enteros diferentes, construir un algoritmo para escribir estos números en forma descendente. Este es un ejemplo de los algoritmos conocidos como de Lógica Pura, ya que poseen muchas decisiones y muchas bifurcaciones.

Salida: A, B y C ordenados descendentemente.

Entradas: A, B y C.

La dinámica del problema es comparar dos números a la vez para conocer cual es el mayor, donde el Flujograma será así:

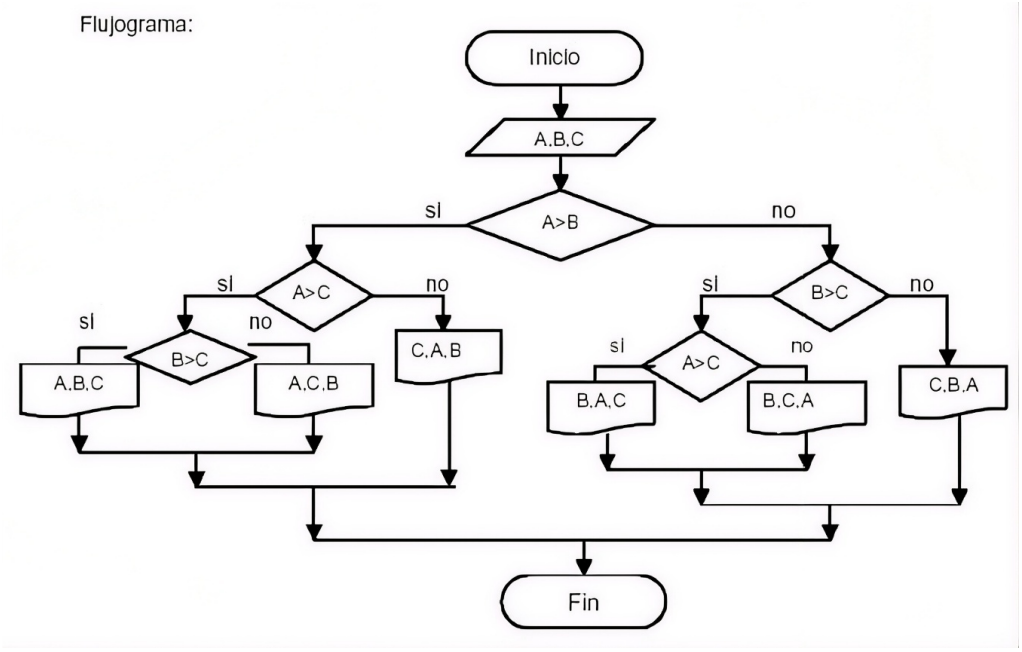
7.2.4 Estructura Múltiple (CASE/SWITCH)

Esta estructura evalúa una variable o expresión y la compara con una lista de casos posibles. Dependiendo del valor coincidente, ejecuta un bloque de código específico. Es más clara y eficiente que una serie de **SI-SINO** anidados cuando las decisiones dependen de múltiples valores discretos de una misma variable.

Ejemplo:

ISBN: 978-631-6960-03-0

Figura 7.13: Ejemplo de Estructura Selectiva Anidada - Diagrama de Flujo



Elaboración propia.

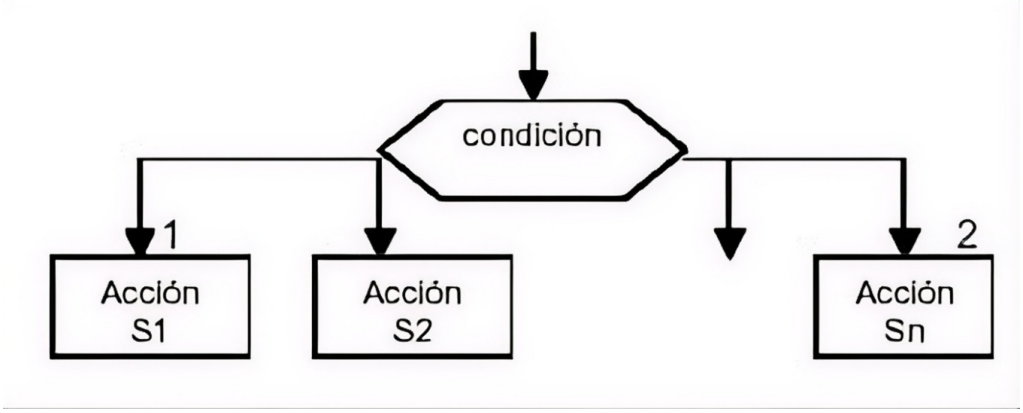
Figura 7.14: Estructura Selectiva Múltiple SWITCH

```
SEGUN (opcion) HACER
    CASO 1:
        Escribir "Opción 1 seleccionada."
    CASO 2:
        Escribir "Opción 2 seleccionada."
    CASO 3:
        Escribir "Opción 3 seleccionada."
    DE OTRO MODO:
        Escribir "Opción no válida."
FINSEGUN
```

Elaboración propia.

ISBN: 978-631-6960-03-0

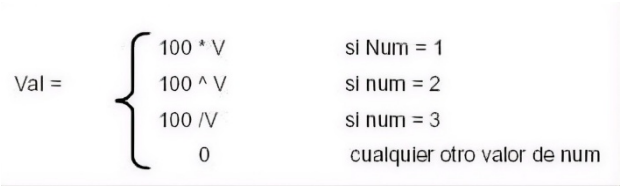
Figura 7.15: Diagrama de Selección Múltiple



Elaboración propia.

Diseñar un algoritmo tal que, dados como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

Figura 7.16: Condiciones del ejemplo para selección Múltiple



Elaboración propia.

7.3 Estructuras Repetitivas

Son operaciones que se deben ejecutar un número repetido de veces. El conjunto de instrucciones que se ejecuta repetidamente cierto número de veces, se llama Ciclo, Bucle o Lazo.

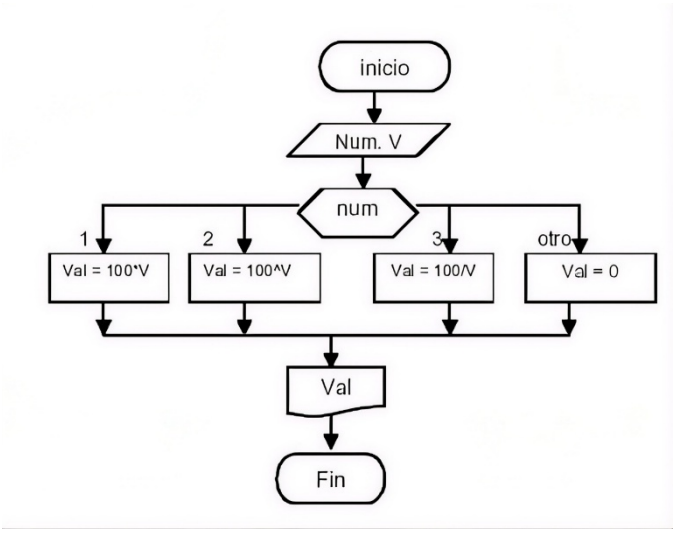
Iteración es cada una de las diferentes pasadas o ejecuciones de todas las instrucciones contenidas en el bucle.

Fases de un Programa Cíclico:

- Entrada de datos e instrucciones previas
- Lazo o bucle

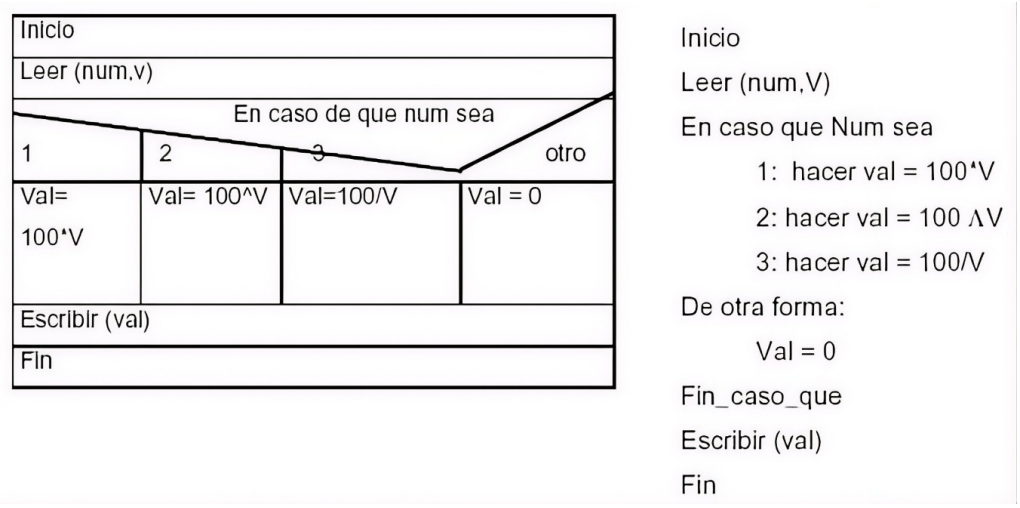
ISBN: 978-631-6960-03-0

Figura 7.17: Diagrama de flujo del ejemplo para Selección Múltiple



Elaboración propia.

Figura 7.18: Diagrama N-S del ejemplo de Selección Múltiple



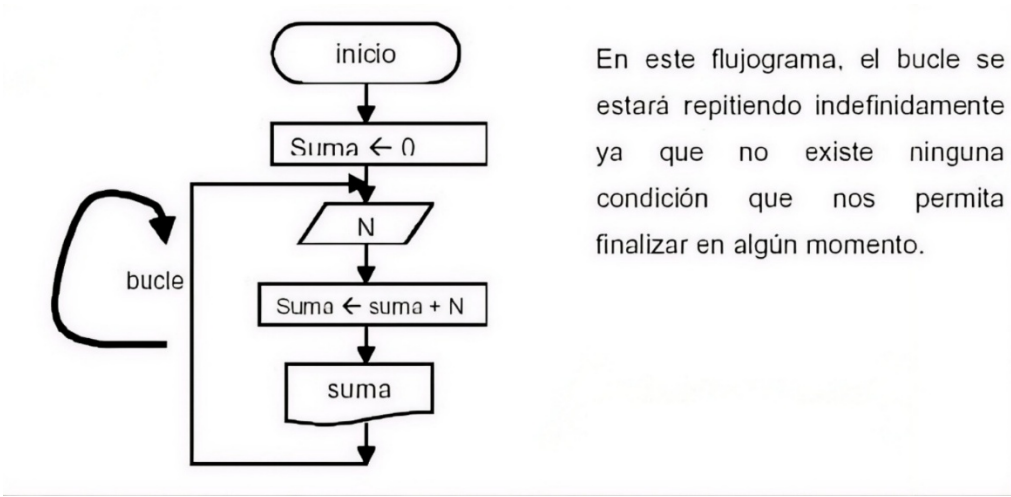
Elaboración propia.

ISBN: 978-631-6960-03-0

- Instrucciones finales o resto del proceso
- Salida de resultado

Ejemplo de bucle infinito:

Figura 7.19: Ejemplo de Buble Infinito



Elaboración propia.

En el flujograma anterior, observa que la flecha que se regresa hacia arriba nos está indicando que hay que volver a evaluar la expresión. En ese caso como el bucle es infinito, no se tiene una condición para terminar y se estará haciendo siempre. En el siguiente ejemplo, ya se agregó una condición, la cual nos permitirá finalizar la ejecución del bucle en el caso en que la condición se cumpla.

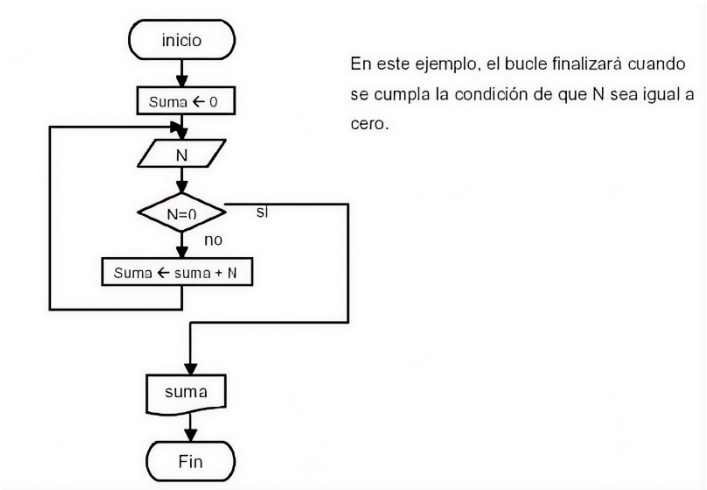
Ejemplo de bucle finito:

A continuación, te muestro tres diseños de estructuras cíclicas: las independientes son cuando los bucles se realizan uno primero hasta que se cumple la condición y solo en ese caso se entra al bucle B. En los ciclos anidados, al entrar a una estructura de repetición, dentro de ella se encuentra otra. La más interna se termina de realizar y se continúa con la externa hasta que la condición se cumple.

En los bucles cruzados, los cuales no son convenientes de utilizar, se tiene que iniciamos un bucle y no se ha terminado cuando empezamos otro, luego utilizamos estructuras goto (saltos) para pasar al bucle externo y se quedan entrelazados. Esto puede ocasionar que el programa pierda el control de cual proceso se está ejecutando

ISBN: 978-631-6960-03-0

Figura 7.20: Ejemplo de Bucle Finito



Elaboración propia.

y podamos obtener resultados erróneos. Veamos gráficamente el diseño de estas tres formas cíclicas:

7.3.1 Estructura PARA (FOR)

Esta estructura se utiliza cuando se conoce de antemano el número exacto de repeticiones. Un contador controla el ciclo, que se ejecuta una vez por cada valor en un rango definido. La variable de control se incrementa (o decrementa) automáticamente en cada iteración.

7.3.2 Estructura MIENTRAS (WHILE)

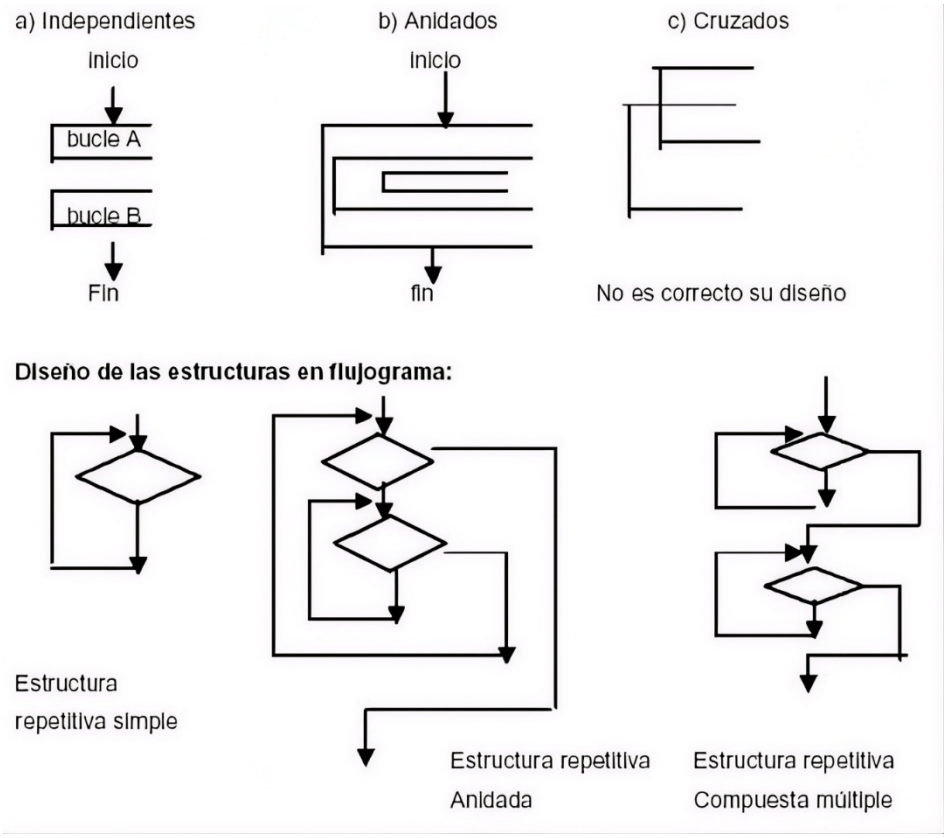
Este ciclo repite un bloque de instrucciones mientras una condición sea verdadera. La condición se evalúa antes de cada iteración. Si es falsa desde el principio, el bloque no se ejecuta ninguna vez. Es crucial que algo dentro del bucle pueda modificar la condición, o se creará un ciclo infinito.

7.3.3 Estructura REPETIR (DO-WHILE)

Similar al MIENTRAS, pero la condición se evalúa después de ejecutar el bloque. Esto garantiza que el bloque se ejecute al menos una vez. El ciclo continúa hasta que la condición se vuelva verdadera (o mientras sea falsa, según la sintaxis del lenguaje).

ISBN: 978-631-6960-03-0

Figura 7.21: Diagramas que se utiliza en las diferentes estructuras repetitivas



Elaboración propia.

Figura 7.22: Estructura Repetitiva Para – FOR

```
1  // Imprimir números del 1 al 5
2  PARA i DESDE 1 HASTA 5 CON PASO 1 HACER
3      Escribir i
4  FINPARA
```

Elaboración propia.

ISBN: 978-631-6960-03-0

Figura 7.23: Estructura Repetitiva Mientras – WHILE

```
MIENTRAS (intentos < 3) HACER
    Escribir "Ingrese su contraseña:"
    Leer password
    intentos ← intentos + 1
FINMIENTRAS
```

Elaboración propia.

Figura 7.24: Estructura Repetitiva REPETIR - DO WHILE

```
REPETIR
    Escribir "Ingrese un número positivo:"
    Leer numero
HASTA QUE (numero > 0) // Se repite hasta que se cumpla la condición.
```

Elaboración propia.

Tabla 7.1: Comparación entre estructuras repetitivas

Estructura	Evaluación Condición	Iteraciones Mínimas	Uso Ideal
PARA (FOR)	Al inicio de cada ciclo (implícita en el contador).	0 (si el rango es inválido, ej. DESDE 5 HASTA 1).	Cuando se conoce el número exacto de repeticiones.
MIENTRAS (WHILE)	Al inicio de cada ciclo.	0.	Cuando las repeticiones dependen de una condición que puede ser falsa inicialmente.
REPETIR (DO-WHILE)	Al final de cada ciclo.	1.	Cuando el bloque debe ejecutarse al menos una vez (ej., menús, validación).

Elaboración propia.

ISBN: 978-631-6960-03-0

7.4 Ejemplos Integrados

7.4.1 Cálculo de factorial

Ilustra el uso de un bucle PARA para un cálculo iterativo.

Figura 7.25: Algoritmo de cálculo de factorial de un número

```
1  Algoritmo Factorial
2      Definir n, i, fact Como Entero
3      fact ← 1
4      Escribir "Ingrese un número:"
5      Leer n
6      PARA i DESDE 1 HASTA n HACER
7          fact ← fact * i
8      FINPARA
9      Escribir "El factorial de ", n, " es
10 FinAlgoritmo
```

Elaboración propia.

7.4.2 Validación de entrada con centinela

Muestra el uso de un bucle MIENTRAS con un valor centinela (-1) para controlar la repetición.

7.4.3 Menú interactivo

Combina REPETIR para garantizar al menos una visualización del menú, SEGUN para las opciones y SI para validar salida.

Este capítulo presentó las herramientas que dotan de lógica y flexibilidad a un programa. Se inició con la estructura secuencial como base. Luego, se exploraron las estructuras selectivas (SI, SI-SINO, anidadas y SEGUN), que permiten tomar decisiones y bifurcar el flujo de ejecución. Posteriormente, se introdujeron las estructuras repetitivas

ISBN: 978-631-6960-03-0

Figura 7.26: Algoritmo de una Suma centinela (Iterativa)

```
Algoritmo SumaCentinela
    Definir numero, suma Como Entero
    suma ← 0
    Escribir "Ingrese números a sumar (-1 para terminar):
    Leer numero
    MIENTRAS (numero ≠ -1) HACER
        suma ← suma + numero
        Leer numero
    FINMIENTRAS
    Escribir "La suma total es: ", suma
FinAlgoritmo
```

Elaboración propia.

Figura 7.27: Algoritmo de Menú Interactivo

```
Algoritmo MenuInteractivo
    Definir opcion Como Entero
    REPETIR
        Escribir "1. Opción A"
        Escribir "2. Opción B"
        Escribir "3. Salir"
        Leer opcion
        SEGUN (opcion) HACER
            CASO 1: Escribir "Ejecutando A..."
            CASO 2: Escribir "Ejecutando B..."
            CASO 3: Escribir "Saliendo..."
            DE OTRO MODO: Escribir "Opción errónea."
        FINSEGUN
    HASTA QUE (opcion == 3)
FinAlgoritmo
```

Elaboración propia.

ISBN: 978-631-6960-03-0

(PARA, MIENTRAS, REPETIR), que permiten automatizar tareas mediante ciclos controlados por condiciones o contadores. Finalmente, se integraron estos conceptos en ejemplos prácticos como el cálculo factorial, la validación con centinela y la creación de un menú interactivo. El dominio de estas tres categorías de control de flujo es absolutamente esencial para pasar de escribir secuencias simples a desarrollar programas que puedan resolver problemas del mundo real de manera eficiente y elegante.

ISBN: 978-631-6960-03-0

CAPÍTULO 8

ESTRUCTURAS DE DATOS: ARREGLOS Y CADENAS

Hasta ahora hemos trabajado con variables que almacenan un único valor (datos simples). Sin embargo, muchos problemas requieren manejar colecciones de datos relacionados, como una lista de calificaciones, una planilla de empleados o una tabla de temperaturas. Las estructuras de datos permiten organizar y gestionar estos conjuntos de manera eficiente. Este capítulo se centra en dos estructuras fundamentales: los arreglos (unidimensionales y bidimensionales) y las cadenas de caracteres, que son esenciales para el procesamiento de información en volumen.

8.1 Datos estructurados

Estructura de Datos es una colección de datos que se caracterizan por su organización y las operaciones que se definen en ella. Los datos de tipo estándar pueden ser organizados en diferentes estructuras de datos: estáticas y dinámicas.

8.2 Estructura de Datos Estáticas

Son aquellas en las que el espacio ocupado en memoria se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa. Corresponden a este tipo los arrays y registros.

8.3 Estructuras de Datos Dinámicas

Son aquellas en las que el espacio ocupado en memoria puede ser modificado en tiempo de ejecución. Corresponden a este tipo las listas, árboles y grafos; Estas estructuras no son soportadas en todos los lenguajes. La elección de la estructura de datos idónea dependerá de la naturaleza del problema a resolver y, en menor medida, del lenguaje. Las estructuras de datos tienen en común que un identificador, nombre, puede representar a múltiples datos individuales.

ISBN: 978-631-6960-03-0

8.4 Arrays

Un arreglo (array) es una colección de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común. Para referirse a un determinado elemento de un array se deberá utilizar un índice, que especifique su posición relativa en el array. Un arreglo es una colección finita, homogénea y ordenada de elementos. Finita: Todo arreglo tiene un límite; es decir, debe determinarse cuál será el número máximo de elementos que podrán formar parte del arreglo. Homogénea: Todos los elementos del arreglo deben ser del mismo tipo. Ordenada: Se puede determinar cuál es el primer elemento, el segundo, el tercero,.... y el n-esimo elemento.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así se tienen los:

- Unidimensionales (vectores)
- Bidimensionales (tablas o matrices)
- Multidimensionales (tres o mas dimensiones)

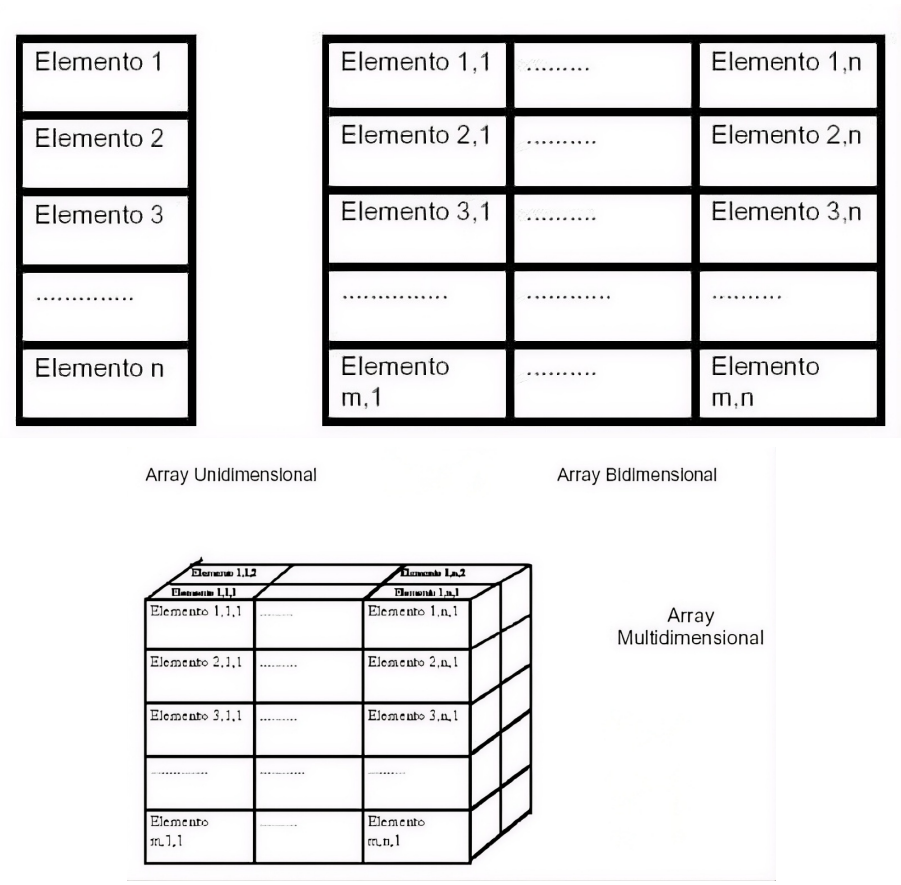
8.5 Arreglos Unidimensionales

Están formados por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos. Al declarar un arreglo, se debe inicializar sus elementos antes de utilizarlos. Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener. Por ejemplo, las siguientes instrucciones declaran tres arreglos distintos:

Para acceder a valores específicos del arreglo, use un valor de índice que apunte al elemento deseado. Por ejemplo, para acceder al primer elemento del arreglo calificaciones debe utilizar el valor de índice 0 (calificaciones[0]). Los programas en C++ siempre indican el primer elemento de un arreglo con 0 y el último con un valor menor en una unidad al tamaño del arreglo.

ISBN: 978-631-6960-03-0

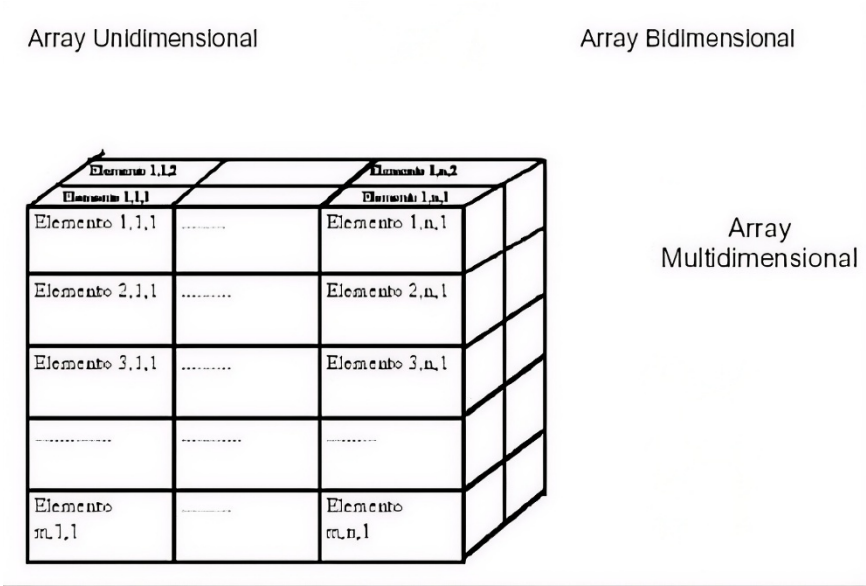
Figura 8.1: Arrays Unidimensionales y Bidimensionales



Elaboración propia.

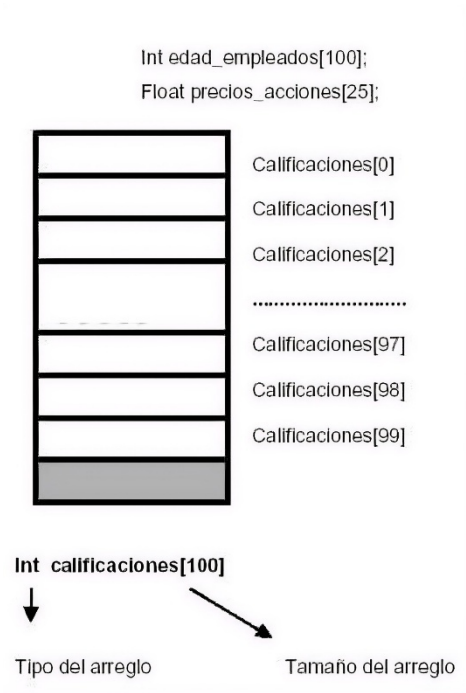
ISBN: 978-631-6960-03-0

Figura 8.2: Arrays Multidimensional



Elaboración propia.

Figura 8.3: Declaración de 3 arreglos distintos



Elaboración propia.

ISBN: 978-631-6960-03-0

8.5.1 Definición y declaración

Se declara especificando el tipo de datos de sus elementos, un nombre y su tamaño (el número de elementos que puede contener). La posición del primer elemento suele ser el índice 0 o 1, dependiendo del lenguaje.

Pseudocódigo: Definir calificaciones[50] Como Entero

C++: int calificaciones[50];

Esto reserva en memoria 50 espacios consecutivos para almacenar valores enteros, accesibles como calificaciones[0], calificaciones[1], ..., calificaciones[49].

8.5.2 Inicialización y asignación

Se puede inicializar al declararlo o asignar valores posteriormente.

Inicialización:

Definir valores[5] Como Entero = {10, 20, 30, 40, 50}

Asignación individual:

calificaciones[0] <- 85

calificaciones[1] <- 90

Asignación masiva (con bucle):

Figura 8.4: Asignación masiva de valores a un vector

```
PARA i DESDE 0 HASTA 49 HACER
    calificaciones[i] ← 0
FINPARA
```

Elaboración propia.

8.5.3 Partes de un arreglo

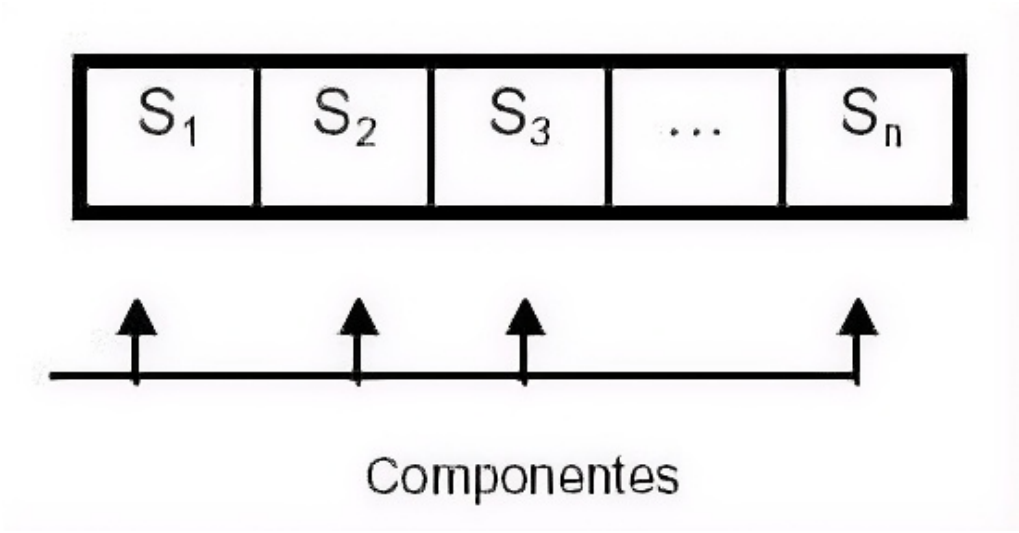
Los componentes. Hacen referencia a los elementos que forman el arreglo, es decir, a los valores que se almacenan en cada una de las casillas de este. Los índices. Permiten hacer referencia a los componentes del arreglo en forma individual, especifican cuantos elementos tendrá el arreglo y, además, de qué modo podrán acceder a esos componentes.

Ejemplos:

Sea arre un arreglo de 70 elementos enteros con índices enteros. Su representación nos queda:

ISBN: 978-631-6960-03-0

Figura 8.5: Componentes de un Arreglo



Elaboración propia.

Figura 8.6: Representación gráfica de un arreglo de 70 elementos

Elaboración propia.

ISBN: 978-631-6960-03-0

8.5.4 Operaciones básicas

Las operaciones más comunes con vectores incluyen:

- **Recorrido (acceso secuencial):** Visitar cada elemento, generalmente con un bucle PARA.
- **Búsqueda:** Determinar si un valor específico existe en el arreglo y en qué posición.
- **Inserción y eliminación:** Agregar o quitar elementos, lo que puede requerir desplazar los demás (es más complejo en arreglos de tamaño fijo).
- **Ordenación:** Organizar los elementos en un orden específico (ascendente o descendente).

8.6 Arreglos Bidimensionales (Matrices)

Un arreglo bidimensional, o matriz, es una estructura de datos homogénea organizada en filas y columnas, formando una tabla rectangular. Es conceptualmente un .ªrreglo de arreglos"donde cada fila es a su vez un arreglo unidimensional (Cairó, 1995). Es un conjunto de datos homogéneo, finito y ordenado, donde se hace referencia a cada elemento por medio de dos índices. El primero se utiliza para los renglones (filas) y el segundo para las columnas. También puede definirse como un arreglo de arreglos. Internamente en memoria se reservan MxN posiciones consecutivas para almacenar todos los elementos del arreglo.

8.6.1 Definición y declaración

Se declara especificando el tipo, el nombre y las dimensiones (número de filas y columnas).

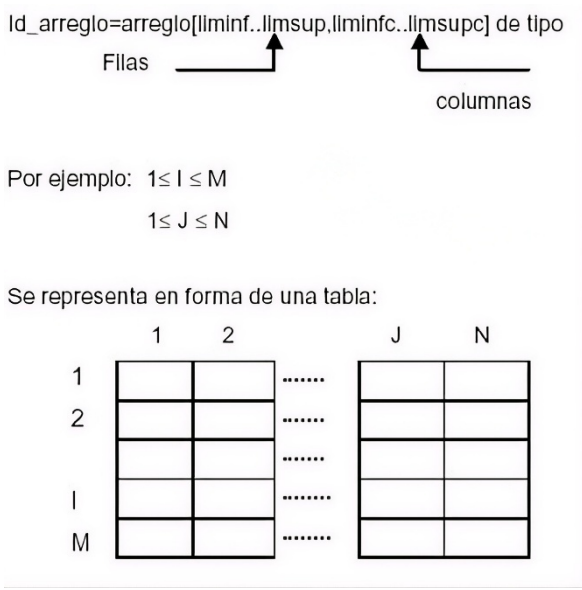
Pseudocódigo: Definir matriz[4,5] Como Real

C++: float matriz[4][5];

Esto crea una tabla de 4 filas y 5 columnas, con 20 elementos en total. Se accede con dos índices: matriz[fila][columna] (ej., matriz[2][3]).

ISBN: 978-631-6960-03-0

Figura 8.7: Declaración de una matriz



Elaboración propia.

8.6.2 Recorrido por filas y columnas

Para procesar todos los elementos de una matriz se utilizan bucles anidados. El orden (filas o columnas primeros) es importante según la operación.

Recorrido por filas (el más común):

Figura 8.8: Como se recorre una matriz por filas

```
PARA f DESDE 0 HASTA 3 HACER
    PARA c DESDE 0 HASTA 4 HACER
        Escribir matriz[f,c]
    FINPARA
FINPARA
```

Elaboración propia.

El recorrido por columnas se hace de manera similar, invirtiendo el sentido de los índices.

El número de elementos que contendrá una fila viene dado por U1-L1+1 (Valor mayor - valor menor +1). Igualmente, el número de elementos para la columna es U2-L2+1. Asm, el número total de elementos de la tabla es (U2-L2+1)*(U1-L1+1)

ISBN: 978-631-6960-03-0

Figura 8.9: Como se recorre una matriz por columnas

```
PARA c DESDE 0 HASTA 4 HACER
    PARA f DESDE 0 HASTA 3 HACER
        ESCRIBIR matriz[f,c]
    FIN PARA
FIN PARA
```

Elaboración propia.

8.6.3 Ejemplos

Un ejemplo clásico es la matriz identidad, donde los elementos de la diagonal principal son 1 y el resto son 0.

8.7 Cadenas de Caracteres

Una cadena de caracteres es una secuencia ordenada de símbolos (letras, dígitos, signos) que se almacena como un arreglo unidimensional de tipo carácter, terminado generalmente por un carácter especial nulo (\0) que marca el fin de la cadena (Gottfried, 1997). Una cadena es un conjunto de caracteres incluido el espacio en blanco, que se almacena en un area contigua de la memoria central. La longitud de una cadena es el nzmero de caracteres que contiene. Una cadena vacma es la que no tiene ningzn caracter. Una constante de tipo cadena es un conjunto de caracteres validos encerrados entre comillas. Una variable de cadena es aquella cuyo contenido es una cadena de caracteres. El zltimo caracter de la cadena marca el fin de la cadena.

Las variables de cadena se dividen en:

- **Estáticas:** Su longitud se define antes de ejecutar el programa y no puede cambiarse a lo largo de este.
- **Semiestaticas:** Su longitud puede variar durante la ejecución del programa, pero sin sobrepasar un límite máximo declarado al principio.
- **Dinámicas:** Su longitud puede variar sin limitación dentro del programa.

ISBN: 978-631-6960-03-0

Figura 8.10: Ejemplo de crear una Matriz de Identidad

```
1  Algoritmo MatrizIdentidad
2      Definir identidad Como Entero
3      Dimension identidad[3,3]
4
5      PARA i DESDE 0 HASTA 2 HACER
6          PARA j DESDE 0 HASTA 2 HACER
7              SI (i == j) ENTONCES
8                  identidad[i,j] ← 1
9              SINO
10                 identidad[i,j] ← 0
11             FINSI
12         FINPARA
13     FINPARA
14 FinAlgoritmo
```

Elaboración propia.

ISBN: 978-631-6960-03-0

8.7.1 Definición y tipos

Se pueden definir como arreglos de caracteres o usando un tipo de dato string si el lenguaje lo soporta.

Como arreglo: Definir nombre[30] Como Caracter (almacena hasta 29 caracteres + \0).

Como tipo string: Definir nombre Como Cadena (el manejo de memoria suele ser dinámico).

8.7.2 Operaciones básicas

Las operaciones más comunes con cadenas incluyen:

Asignación:

nombre <- "Rosa"

Concatenación: Unir dos o más cadenas.

nombreCompleto <- nombre + + apellido

Longitud: Obtener el número de caracteres de la cadena.

long <- Longitud(saludo)

Comparación: Determinar si dos cadenas son iguales, o cuál es mayor alfabéticamente (según el código ASCII/Unicode).

SI (pass1 == pass2) ENTONCES ...

Extracción de subcadenas: Obtener una parte de la cadena.

sub <- SubCadena(frase, 1, 5) // Extrae desde la posición 1, 5 caracteres.

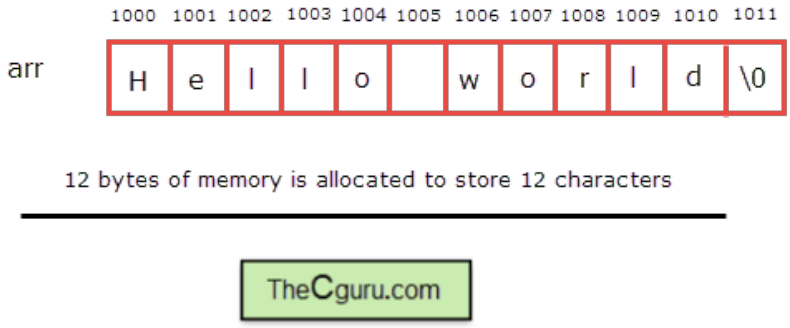
Búsqueda: Encontrar la posición de un carácter o subcadena dentro de otra.

pos <- Encontrar("mundial", frase)

Este capítulo amplió significativamente nuestra capacidad para modelar datos al introducir estructuras de datos compuestas. Se definieron los arreglos unidimensionales (vectores) como colecciones ordenadas e indexadas, detallando su declaración, inicialización y operaciones básicas de recorrido. Luego, se extendió el concepto a los arreglos bidimensionales (matrices), fundamentales para representar datos tabulares mediante filas y columnas, y se explicó su recorrido con bucles anidados. Finalmente, se estudió un caso especial y omnipresente: las cadenas de caracteres, tratadas como arreglos de caracteres con operaciones específicas como concatenación, comparación y búsqueda. El dominio de arreglos y cadenas es crucial para manejar conjuntos de datos en problemas reales, desde procesar listas de estudiantes hasta manipular texto e imágenes, sentando

ISBN: 978-631-6960-03-0

Figura 8.11: Representación en memoria de la cadena "Hello worldçomo arreglo de caracteres



Fuente: (OverIQ, 2020)

las bases para estructuras de datos más complejas.

ISBN: 978-631-6960-03-0

CAPÍTULO 9

PROGRAMACIÓN MODULAR Y ALGORITMOS DE ORDENAMIENTO

Habíamos visto la programación estructurada que permite la escritura de programas fáciles de leer y modificar. En esta programación, el flujo lógico se gobierna por las estructuras de control básicas vista hasta hoy: secuenciales, repetitivas y de selección. La programación modular permite la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de resolver y que pueden ser tratados separadamente unos de otros.

Gracias a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurándose sus errores antes de su inclusión en el programa principal y almacenarse para su posterior utilización cuantas veces se precise.

9.1 Módulo

Uno de los elementos principales de programación utilizados en la representación de cada módulo es la subrutina. Una subrutina es un conjunto de instrucciones de cómputo que realizan una tarea. Un programa principal llama a estos módulos a medida que se necesitan. Un módulo es un segmento, rutina, subrutina, subalgoritmo o procedimiento, que puede definirse dentro de un algoritmo con el fin de ejecutar una tarea específica y puede ser llamado o invocado desde el algoritmo principal cuando sea necesario. Los módulos son independientes en el sentido de que ningún módulo puede tener acceso directo a cualquier otro módulo, con excepción del módulo al que llama y sus propios submódulos. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera a ellos el control. Los módulos tienen una entrada y una salida. Se pueden tomar decisiones dentro de un módulo que tenga repercusión en todo el flujo, pero el salto debe ser únicamente hacia el programa principal. Al descomponer un programa en módulos independientes más simples se conoce también como el método de "Divide y vencerás".

ISBN: 978-631-6960-03-0

9.1.1 ¿Cuándo es útil la modularización?

Este enfoque de segmentación o modularización es útil en dos casos:

1. Cuando existe un grupo de instrucciones o una tarea específica que deba ejecutarse en más de una ocasión.

2. Cuando un problema es complejo o extenso, la solución se divide o segmenta en módulos que ejecutan partes o tareas específicas.

9.1.2 Ventajas de la Programación Modular

Como los módulos son independientes, el desarrollo de un programa se puede efectuar con mayor facilidad, dado que cada módulo se puede crear aisladamente y varios programadores podrán trabajar simultáneamente en la confección de un algoritmo, repartiéndose las distintas partes de este. Se podrá modificar un módulo sin afectar a los demás Las tareas, subalgoritmos, sólo se escribirán una vez, aunque se necesiten en distintas ocasiones a lo largo del algoritmo. El uso de módulos facilita la proyección y la comprensión de la lógica subyacente para el programador y el usuario. Aumenta la facilidad de depuración y búsqueda de errores en un programa ya que éstos se pueden aislar fácilmente.

El mantenimiento y la modificación de la programación se facilitan. Los módulos reciben diferentes nombres:

- Funciones en C, C++
- Subrutinas en Basic
- Procedimientos y funciones en Pascal
- Subrutinas en Fortran
- Secciones en Cobol.

9.1.3 Desarrollar programas de forma modular

Significa que pueden identificarse las principales tareas a realizar por el programa y que se pueden diseñar y probar procedimientos individuales para estas tareas. Por ejemplo: Qué transacciones se le hacen a una cuenta de ahorros?

Transacciones:

ISBN: 978-631-6960-03-0

- Depósito (cheque y efectivo)
- Intereses
- Retiro
- Estado de cuenta
- Cambio de libreta

9.1.4 Procedimientos y funciones

Ambos son los bloques de construcción de la programación modular. Un procedimiento (a veces llamado subrutina) es un módulo que ejecuta una tarea pero no devuelve un valor al punto donde fue llamado. Su propósito es realizar acciones, como mostrar un menú o ordenar un arreglo. Una función, en cambio, es un módulo que realiza un cálculo o determinación y devuelve un único valor a quien lo invoca. Se utiliza en expresiones, como resultado <- calcularPromedio(calificaciones).

9.2 Paso de Parámetros

Los parámetros son mecanismos para comunicar datos entre el programa principal (o un módulo) y un procedimiento o función. Definen qué información recibe el módulo para trabajar. Existen dos formas fundamentales de pasar esta información, con implicaciones muy diferentes.

9.2.1 Paso por valor

Cuando un parámetro se pasa por valor, el módulo recibe una copia del dato original. Cualquier modificación que el módulo haga sobre este parámetro solo afecta a la copia, no al dato original en el programa que hizo la llamada. Es útil cuando se quiere usar el valor de una variable sin riesgo de alterarla accidentalmente.

9.2.2 Paso por referencia

Cuando un parámetro se pasa por referencia, el módulo recibe la dirección en memoria (una referencia) del dato original. Esto significa que el módulo trabaja

ISBN: 978-631-6960-03-0

directamente con la variable original. Cualquier cambio que realice el módulo en el parámetro se reflejará directamente en la variable original del llamante. Se usa cuando el módulo necesita modificar el valor de una variable o devolver múltiples resultados.

9.3 Variables Locales y Globales

El ámbito de una variable determina en qué partes del programa es visible y puede ser utilizada. Comprender esta distinción es clave para evitar errores y escribir código modular robusto.

9.3.1 Tiempo de vida de los datos

Según el lugar donde son declaradas puede haber dos tipos de variables.

Globales: Las variables permanecen activas durante todo el programa. Se crean al iniciarse éste y se destruyen de la memoria al finalizar. Pueden ser utilizadas en cualquier procedimiento o función.

Locales: Las variables son creadas cuando el programa llega a la función o procedimiento en la que están definidas. Al finalizar la función o el procedimiento, desaparecen de la memoria. Si dos variables, una global y una local, tienen el mismo nombre, la local prevalecerá sobre la global dentro del módulo en que ha sido declarada. Dos variables locales pueden tener el mismo nombre siempre que estén declaradas en funciones o procedimientos diferentes.

9.3.2 Ámbito de las variables

Una variable global se declara fuera de cualquier módulo (usualmente al inicio del programa) y es accesible desde cualquier parte del código, incluyendo todos los procedimientos y funciones. Una variable local se declara dentro de un módulo específico y solo existe y es accesible dentro de ese módulo. Se crea cuando el módulo se ejecuta y se destruye cuando termina.

9.3.3 Buenas prácticas

Se recomienda minimizar el uso de variables globales, ya que cualquier módulo puede modificarlas, lo que dificulta rastrear errores y comprender el flujo de datos. El uso preferente de variables locales y el paso de parámetros hace que los módulos sean

ISBN: 978-631-6960-03-0

más independientes, reutilizables y predecibles. Los parámetros definen explícitamente la interfaz de comunicación del módulo.

9.4 Procedimientos

Son subprogramas, es decir, módulos que forman parte de un programa y realizan una tarea específica. Un procedimiento puede tener sus propias variables que se declaran en la sección var del propio procedimiento. Estas se llaman variables locales. La casilla de memoria para estas variables se crea cada vez que el procedimiento es llamado y se borran al salir del mismo. Así, las variables locales para un procedimiento solo se pueden usar en el cuerpo del procedimiento y no en el cuerpo principal del programa.

9.5 Funciones

La función es una estructura autónoma similar a los módulos. La diferencia radica en que la función se usa para devolver un solo valor de un tipo de dato simple a su punto de referencia. La función se relaciona especificando su nombre en una expresión, como si fuera una variable ordinaria de tipo simple. Las funciones se dividen en estándares y definidas por el usuario.

- **Estándar:** Son funciones proporcionadas por cualquier lenguaje de programación de alto nivel, y se dividen en aritméticas y alfabéticas.
- **Definidas por el usuario:** son funciones que puede definirlas el programador con el propósito de ejecutar alguna función específica, y que por lo general se usan cuando se trata de hacer algún cálculo que será requerido en varias ocasiones en la parte principal del algoritmo.

9.6 Semejanzas entre Procedimientos y Funciones

- La definición de ambos aparece en la sección de subprogramas de la parte de declaraciones de un programa y en ambos casos consiste en una cabecera, una parte de declaraciones una parte de instrucciones.
- Ambos son unidades de programa independientes. Los parámetros, constantes y variables declarados en una función o procedimiento son locales a la función o al

ISBN: 978-631-6960-03-0

procedimiento, solamente son accesibles dentro del subprograma.

- Cuando se llama a una función o a un procedimiento, el número de los parámetros reales debe ser el mismo que el número de los parámetros formales y los tipos de los parámetros reales deben coincidir con los tipos de los correspondientes parámetros formales, con una excepción: se puede asociar un parámetro real de tipo entero con un parámetro formal por valor de tipo real.

9.7 Diferencias entre Procedimientos y Funciones

- Mientras que a un procedimiento se le llama mediante una instrucción de llamada a procedimiento, a una función se la llama usando su nombre en una expresión.
- Puesto que se debe asociar un valor al número de una función, también se le debe asociar un tipo. Por tanto, la cabecera de una función debe incluir un identificador de tipo que especifique el tipo del resultado. Sin embargo, no se asocia ningún valor con el nombre de un procedimiento y, por tanto, tampoco ningún tipo.
- Las funciones normalmente devuelven un único valor a la unidad de programa que la llama. Los procedimientos suelen devolver más de un valor, o pueden no devolver ninguno si solamente realizan alguna tarea, como una operación de salida.
- En los procedimientos, los valores se devuelven a través de parámetros por variable, pero el valor de una función se devuelve mediante la asignación al nombre de la función de dicho valor en la parte de instrucciones de la definición de la función.

9.8 Métodos de Ordenamiento

Uno de los procedimientos más comunes y útiles en el procesamiento de datos, es la clasificación u ordenación de los mismos. Se considera ordenar al proceso de reorganizar un conjunto dado de objetos en una secuencia determinada. Cuando se analiza un método de ordenación, hay que determinar cuántas comparaciones e intercambios se realizan para el caso más favorable, para el caso medio y para el caso más desfavorable.

La colocación en orden de una lista de valores se llama Ordenación. Por ejemplo, se podría disponer una lista de valores numéricos en orden ascendente o descendente, o bien una lista de nombres en orden alfabético. La localización de un elemento de una

ISBN: 978-631-6960-03-0

lista se llama búsqueda. Tal operación se puede hacer de manera más eficiente después de que la lista ha sido ordenada.

Existen varios métodos para ordenamiento, clasificados en tres formas:

1. Intercambio
2. Selección
3. Inserción.

En cada familia se distinguen dos versiones: un método simple y directo, fácil de comprender, pero de escasa eficiencia respecto al tiempo de ejecución, y un método rápido, más sofisticado en su ejecución por la complejidad de las operaciones a realizar, pero mucho más eficiente en cuanto a tiempo de ejecución. En general, para arreglos con pocos elementos, los métodos directos son más eficientes (menor tiempo de ejecución) mientras que para grandes cantidades de datos se deben emplear los llamados métodos rápidos.

9.8.1 Ordenamiento por Intercambio (Burbuja)

El método de intercambio se basa en comparar los elementos del arreglo e intercambiarlos si su posición actual o inicial es contraria inversa a la deseada. Pertenece a este método el de la burbuja clasificado como intercambio directo. Aunque no es muy eficiente para ordenar listas grandes, es fácil de entender y muy adecuado para ordenar una pequeña lista de unos 100 elementos o menos.

Una pasada por la ordenación de burbujeo consiste en un recorrido completo a través del arreglo, en el que se comparan los contenidos de las casillas adyacentes, y se cambian si no están en orden. La ordenación por burbujeo completa consiste en una serie de pasadas ("burbujeo") que termina con una en la que ya no se hacen cambios porque todo está en orden.

Ejemplo:

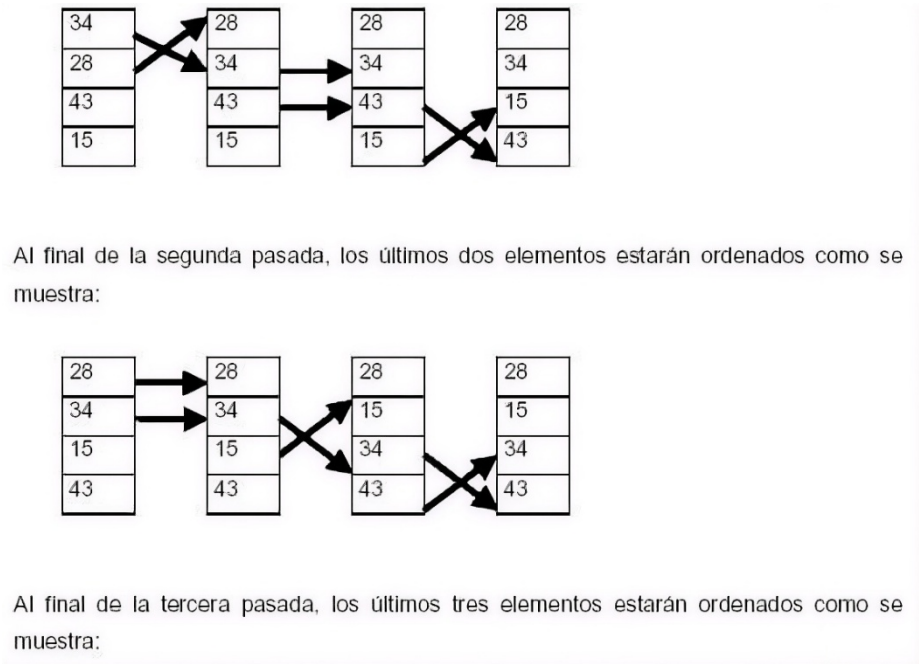
Supóngase que están almacenados cuatro números en un arreglo con casillas de memoria de x[1] a x[4]. Se desea disponer esos números en orden creciente. La primera pasada de la ordenación por burbujeo haría lo siguiente:

- Comparar el contenido de x[1] con el de x[2]; si x[1] contiene el mayor de los números, se intercambian sus contenidos.

ISBN: 978-631-6960-03-0

- Comparar el contenido de x[2] con el de x[3]; e intercambiarlos si fuera necesario.
- Comparar el contenido de x[3] con el de x[4]; e intercambiarlos si fuera necesario.
- Al final de la primera pasada, el mayor de los números estará en x[4].

Figura 9.1: Segunda pasada y Tercera pasada de Intercambio



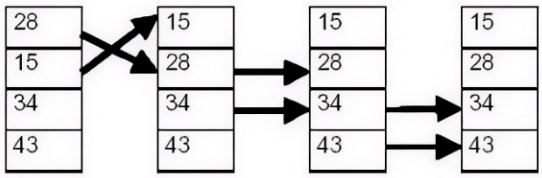
Elaboración propia.

Quicksort

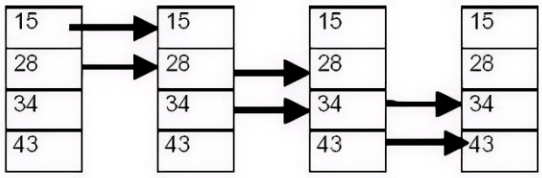
Si bien el método de la burbuja era considerado como el peor método de ordenación simple o menos eficiente, el método Quicksort basa su estrategia en la idea intuitiva de que es más fácil ordenar una gran estructura de datos subdividiéndolas en otras más pequeñas introduciendo un orden relativo entre ellas. En otras palabras, si dividimos el array a ordenar en dos subarrays de forma que los elementos del subarray inferior sean más pequeños que los del subarray superior, y aplicamos el método reiteradamente, al final tendremos el array inicial totalmente ordenado. Existen además otros métodos conocidos, el de ordenación por montículo y el de shell.

ISBN: 978-631-6960-03-0

Figura 9.2: Final pasada para garantizar el orden correcto



Nótese que el arreglo ya quedó ordenado, sin embargo, se hace una pasada más porque la computadora no advierte que el arreglo está en orden hasta que ocurre una pasada sin intercambios.



La ordenación por burbujeo se llama así porque los números más pequeños ascienden como burbujas hasta la parte superior, mientras que los mayores se hunden y caen hasta el fondo. Está garantizado que cada pasada pone el siguiente número más grande en su lugar, aunque pueden colocarse más de ellos en su lugar por casualidad.

Elaboración propia.

ISBN: 978-631-6960-03-0

9.8.2 Ordenamiento por Selección

Los métodos de ordenación por selección se basan en dos principios básicos:

- Seleccionar el elemento más pequeño (o más grande) del arreglo.
- Colocarlo en la posición más baja (o más alta) del arreglo.

A diferencia del método de la burbuja, en este método el elemento más pequeño (o más grande) es el que se coloca en la posición final que le corresponde.

9.8.3 Ordenamiento por Inserción

El fundamento de este método consiste en insertar los elementos no ordenados del arreglo en subarreglos del mismo que ya estén ordenados. Dependiendo del método elegido para encontrar la posición de inserción tendremos distintas versiones del método de inserción.

Tabla 9.1: Comparación de métodos de ordenamiento básicos

Método	Concepto Clave	Ventaja Principal	Desventaja Principal
Burbuja	Intercambiar adyacentes desordenados.	Sencillez conceptual.	Muy lento para muchos datos.
Selección	Seleccionar el mínimo e intercambiarlo.	Menos intercambios que Burbuja.	Tantas comparaciones como Burbuja.
Inserción	Insertar elemento en posición correcta.	Rápido si la lista está casi ordenada.	Lento si los datos están muy desordenados.

Elaboración propia.

9.9 Métodos de Búsqueda

La búsqueda es una operación que tiene por objeto la localización de un elemento dentro de la estructura de datos. A menudo un programador estará trabajando con grandes cantidades de datos almacenados en arreglos y pudiera resultar necesario determinar si un arreglo contiene un valor que coincide con algún valor clave o buscado.

Siendo el array de una dimensión o lista una estructura de acceso directo y a su vez de acceso secuencial, encontramos dos técnicas que utilizan estos dos métodos de acceso, para encontrar elementos dentro de un array: búsqueda lineal y búsqueda binaria.

ISBN: 978-631-6960-03-0

9.9.1 Búsqueda Secuencial

También llamada búsqueda lineal, es la más simple. Consiste en recorrer el arreglo elemento por elemento, desde el principio hasta el final, comparando cada uno con la clave buscada. El proceso termina cuando se encuentra el elemento o se llega al final del arreglo. No requiere que el arreglo esté ordenado, pero es ineficiente para listas grandes, ya que en el peor caso debe revisar todos los elementos.

El método de búsqueda lineal funciona bien con arreglos pequeños o para arreglos no ordenados. Si el arreglo está ordenado, se puede utilizar la técnica de alta velocidad de búsqueda binaria, donde se reduce sucesivamente la operación eliminando repetidas veces la mitad de la lista restante.

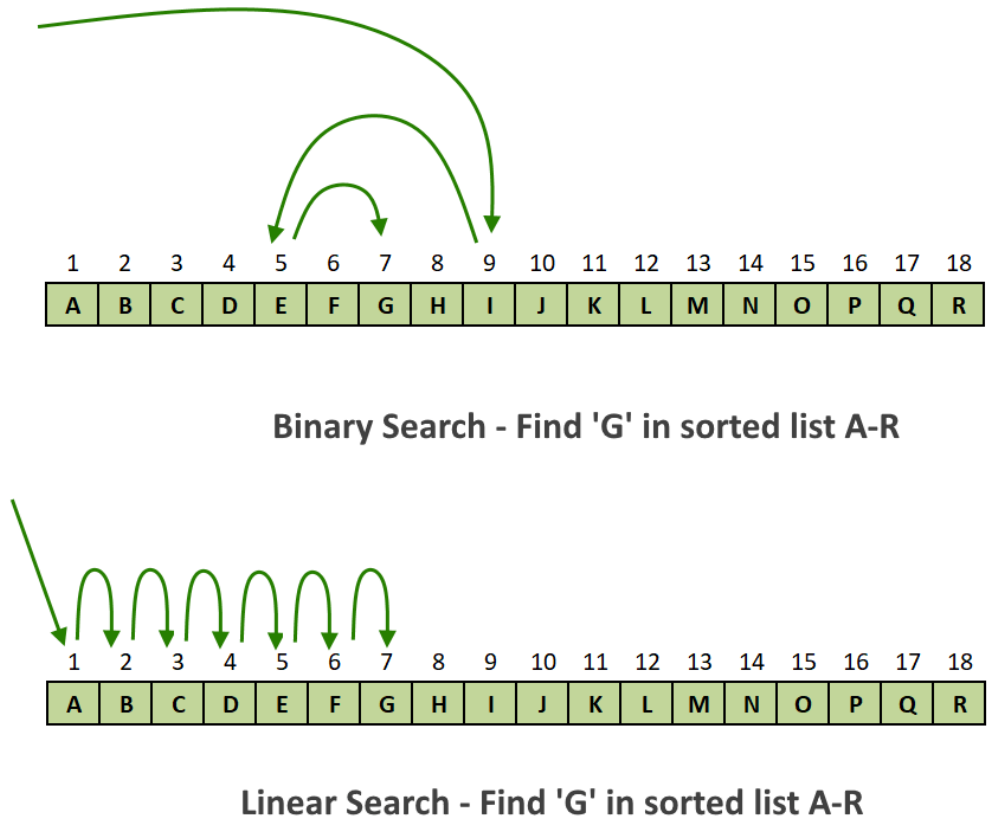
9.9.2 Búsqueda Binaria

La búsqueda binaria es el método más eficiente para encontrar elementos en un arreglo ordenado. El proceso comienza comparando el elemento central del arreglo con el valor buscado. Si ambos coinciden finaliza la búsqueda. Si no ocurre así, el elemento buscado será mayor o menor en sentido estricto que el central del arreglo. Si el elemento buscado es mayor se procede a hacer búsqueda binaria en el subarray superior, si el elemento buscado es menor que el contenido de la casilla central, se debe cambiar el segmento a considerar al segmento que está a la izquierda de tal sitio central.

Este capítulo consolidó dos pilares avanzados de la programación. Primero, se presentó la programación modular como filosofía esencial para gestionar la complejidad, definiendo sus componentes (procedimientos y funciones), los mecanismos de comunicación entre ellos (paso de parámetros por valor y por referencia) y las reglas de visibilidad de datos (variables locales y globales). En segundo lugar, se introdujeron algoritmos clásicos para el procesamiento de arreglos: los métodos de ordenamiento (Burbuja, Selección e Inserción) para organizar datos, y los métodos de búsqueda (Secuencial y Binaria) para recuperarlos eficientemente. Estos conceptos y herramientas no solo elevan la calidad y mantenibilidad del código, sino que también proporcionan las técnicas fundamentales para manipular y extraer valor de los datos, cerrando el ciclo de aprendizaje de los fundamentos de la programación estructurada.

ISBN: 978-631-6960-03-0

Figura 9.3: Diagrama que contrasta la búsqueda secuencial con la búsqueda binaria



Fuente: (tutorialhorizon, 2026)

ISBN: 978-631-6960-03-0

CAPÍTULO 10

MANEJO DE FICHEROS Y ORGANIZACIÓN DE DATOS

En los capítulos anteriores hemos trabajado con datos almacenados en memoria durante la ejecución de un programa. Sin embargo, en aplicaciones reales, es necesario conservar información de manera permanente para su uso en futuras ejecuciones. Los ficheros (archivos) son el mecanismo fundamental para el almacenamiento persistente de datos, permitiendo guardar, recuperar y organizar grandes volúmenes de información en dispositivos de almacenamiento secundario como discos duros o memorias flash. Este capítulo introduce los conceptos esenciales de manejo de ficheros, las diferentes organizaciones de archivos y las operaciones básicas para su manipulación, proporcionando la base necesaria para implementar sistemas de información robustos y eficientes JOYANES.

10.1 Almacenamiento Persistente

La memoria principal (RAM) es volátil, lo que significa que pierde su contenido cuando se apaga la computadora. Para conservar datos entre sesiones de ejecución, es necesario utilizar almacenamiento secundario, donde la información se guarda en forma de archivos. Un archivo es una colección estructurada de datos relacionados, almacenados bajo un nombre único en un sistema de archivos. La programación con archivos permite:

- Mantener registros históricos (ventas, alumnos, inventarios).
- Intercambiar datos entre diferentes programas.
- Recuperar información después de reiniciar el sistema.

ISBN: 978-631-6960-03-0

10.2 Tipos de ficheros

Los archivos pueden clasificarse según dos criterios principales: su formato interno y su método de acceso.

10.2.1 Por formato interno

- **Archivos de texto:** Contienen caracteres legibles por humanos, organizados en líneas terminadas con un carácter especial (como salto de línea). Son editables con cualquier editor de texto simple, pero ocupan más espacio y su procesamiento es más lento. Ejemplo: archivos .TXT, .CSV.
- **Archivos binarios:** Almacenan datos en el mismo formato interno que utiliza la memoria del programa. Son más eficientes en espacio y velocidad, pero no son directamente legibles por humanos. Ejemplo: archivos .DAT, .BIN.

10.2.2 Por método de acceso

- **Acceso secuencial:** Los registros se leen o escriben en orden, uno tras otro, desde el principio hasta el final. Para acceder a un registro en particular, es necesario recorrer todos los anteriores.
- **Acceso directo (aleatorio):** Permite acceder a cualquier registro directamente mediante su posición física o una clave, sin necesidad de leer los registros previos.
- **Acceso indexado:** Combina acceso directo y secuencial mediante el uso de un índice que relaciona claves con posiciones físicas en el archivo.

10.3 Organización de archivos

La organización de un archivo determina cómo se estructuran físicamente los registros en el dispositivo de almacenamiento y cómo se accede a ellos. Las tres organizaciones principales son:

ISBN: 978-631-6960-03-0

10.3.1 Organización secuencial

En esta organización, los registros se almacenan uno después de otro en el orden en que fueron creados. Es la forma más simple y se utiliza cuando:

- El volumen de datos es grande y se procesa en su totalidad (por ejemplo, generación de reportes).
- Las altas y bajas son poco frecuentes.
- No se requiere acceso inmediato a registros específicos.

Ventajas:

- Simple de implementar.
- Eficiente para procesamientos por lotes (batch).

Desventajas:

- Acceso lento a registros individuales.
- Ineficiente para actualizaciones frecuentes.

10.3.2 Organización directa

Cada registro se almacena en una posición calculada mediante una función hash aplicada a una clave (por ejemplo, el DNI). Esta organización permite:

- Acceso inmediato a cualquier registro.
- Actualizaciones rápidas.

Sin embargo, presenta el problema de las colisiones (cuando dos claves diferentes generan la misma posición), que se resuelven mediante técnicas como:

- Encadenamiento: Los registros que colisionan se enlazan en una lista.
- Direccionamiento abierto: Se busca otra posición disponible dentro del archivo.

ISBN: 978-631-6960-03-0

10.3.3 Organización indexada

Utiliza una estructura auxiliar llamada índice, que contiene pares (clave, dirección) para localizar rápidamente los registros en el archivo principal. El índice mismo suele estar ordenado por clave, permitiendo búsquedas binarias rápidas. Es ideal cuando:

- Se necesitan búsquedas frecuentes por diferentes criterios.
- El archivo principal es muy grande.
- Se requiere mantener el archivo principal ordenado físicamente o no.

Tabla 10.1: Clave y Dirección de archivo en la organización indexada

Clave (DNI)	Dirección en archivo
28456789	1204
34567890	0456
45678901	2098

Elaboración propia.

10.4 Operaciones básicas con ficheros

Todas las operaciones con archivos siguen un patrón común, independientemente de su organización:

10.4.1 Apertura del archivo

Antes de cualquier operación, el archivo debe ser abierto en un modo específico:

- **Lectura (R):** Solo permite leer registros existentes.
- **Escritura (W):** Crea un nuevo archivo (borra el anterior si existe).
- **Añadir (A):** Permite agregar nuevos registros al final del archivo.
- **Lectura/Escritura (R+):** Permite ambas operaciones.

ISBN: 978-631-6960-03-0

10.4.2 Lectura y escritura de registros

Una vez abierto, se pueden realizar operaciones de lectura (obtener datos del archivo) y escritura (guardar datos en el archivo). Es crucial verificar el fin de archivo (EOF) durante la lectura para evitar errores.

10.4.3 Cierre del archivo

Finalizadas las operaciones, el archivo debe cerrarse para asegurar que todos los datos se han escrito físicamente en el disco y liberar los recursos del sistema.

10.5 Actualización de archivos

Los archivos deben mantenerse actualizados para reflejar los cambios en la información. Las operaciones básicas de actualización son:

10.5.1 Altas (inserción de nuevos registros)

Dependiendo de la organización:

- **Secuencial:** Se añaden al final o se insertan manteniendo el orden (requiere reescritura).
- **Directa:** Se calcula la posición mediante la función hash.
- **Indexada:** Se añade al archivo principal y se actualiza el índice.

10.5.2 Bajas (eliminación de registros)

- **Secuencial:** Se marcan como eliminados o se regenera el archivo sin ellos.
- **Directa:** Se marca la posición como libre para reutilización.
- **Indexada:** Se elimina la entrada del índice.

10.5.3 Modificaciones (cambios en registros existentes)

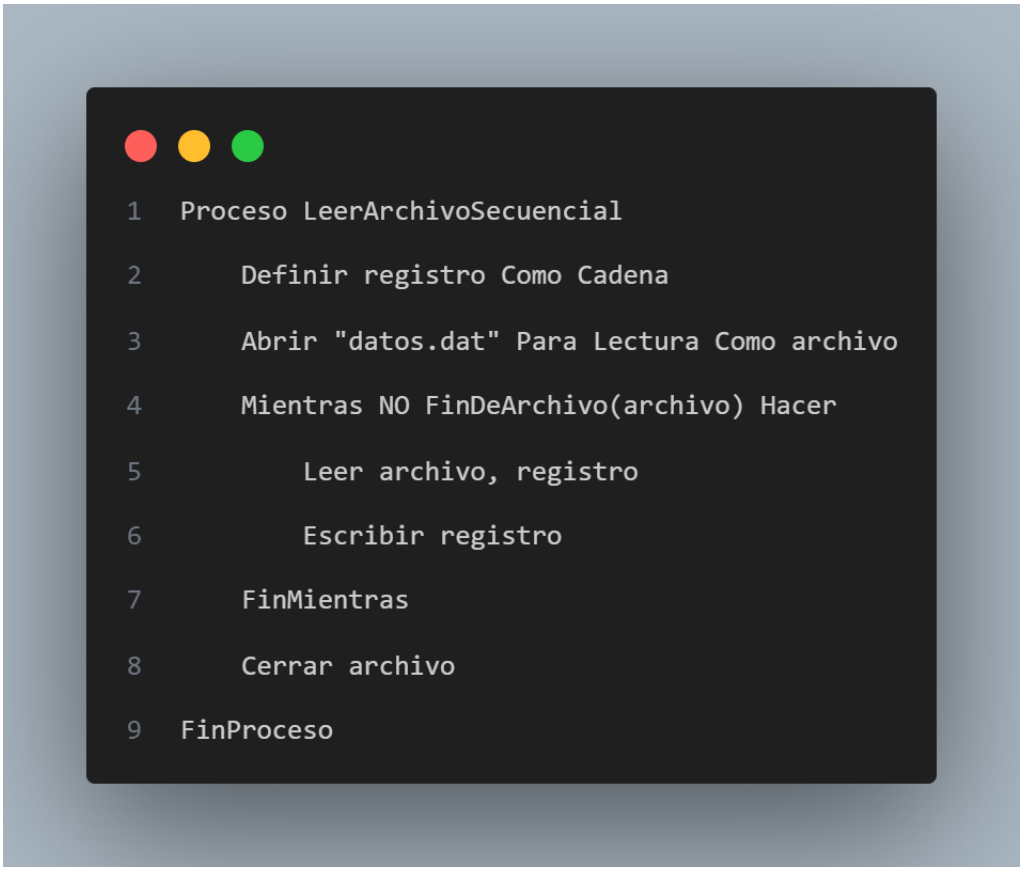
Requieren localizar el registro y sobrescribir sus campos con los nuevos valores.

ISBN: 978-631-6960-03-0

10.6 Ejemplos de algoritmos en pseudocódigo

Lectura secuencial de un archivo

Figura 10.1: Algoritmo para lectura secuencial de un archivo



```
Proceso LeerArchivoSecuencial
    Definir registro Como Cadena
    Abrir "datos.dat" Para Lectura Como archivo
    Mientras NO FinDeArchivo(archivo) Hacer
        Leer archivo, registro
        Escribir registro
    FinMientras
    Cerrar archivo
FinProceso
```

Elaboración propia.

Búsqueda en archivo indexado

Inserción en archivo de organización directa con encadenamiento

ISBN: 978-631-6960-03-0

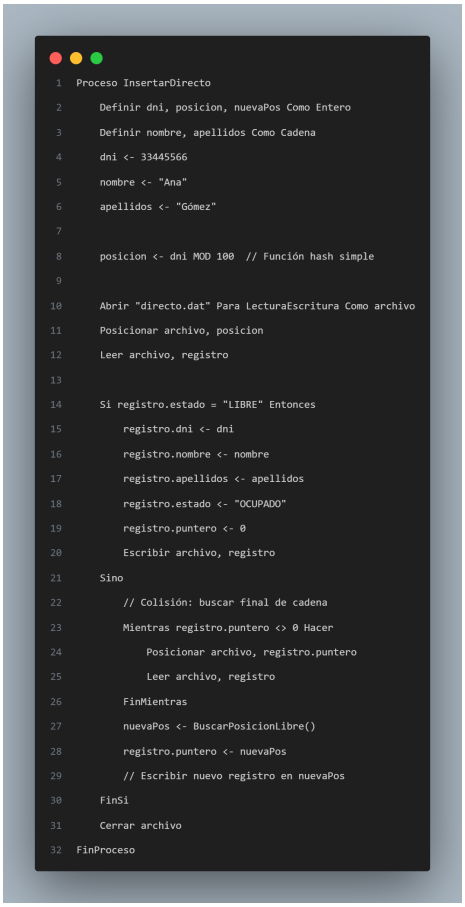
Figura 10.2: Algoritmo de Búsqueda de un archivo indexado

```
Proceso BuscarEnIndexado
    Definir claveBuscada, claveIdx Como Entero
    Definir posicion Como Entero
    claveBuscada <- 45678901
    Abrir "indice.idx" Para Lectura Como idx
    posicion <- -1
    Mientras NO FinDeArchivo(idx) Y posicion = -1 Hacer
        Leer idx, claveIdx, posicion
        Si claveIdx = claveBuscada Entonces
            Escribir "Registro encontrado en posición: ", posicion
        FinSi
    FinMientras
    Si posicion = -1 Entonces
        Escribir "Registro no encontrado"
    FinSi
    Cerrar idx
FinProceso
```

Elaboración propia.

ISBN: 978-631-6960-03-0

Figura 10.3: Algoritmo de la inserción de archivo de organización directa con encadenamiento



```
Proceso InsertarDirecto
    Definir dni, posicion, nuevaPos Como Entero
    Definir nombre, apellidos Como Cadena
    dni <- 33445566
    nombre <- "Ana"
    apellidos <- "Gómez"

    posicion <- dni MOD 100  // Función hash simple

    Abrir "directo.dat" Para LecturaEscritura Como archivo
    Posicionar archivo, posicion
    Leer archivo, registro

    Si registro.estado = "LIBRE" Entonces
        registro.dni <- dni
        registro.nombre <- nombre
        registro.apellidos <- apellidos
        registro.estado <- "OCUPADO"
        registro.puntero <- 0
        Escribir archivo, registro
    Sino
        // Colisión: buscar final de cadena
        Mientras registro.puntero <> 0 Hacer
            Posicionar archivo, registro.puntero
            Leer archivo, registro
        FinMientras
        nuevaPos <- BuscarPosicionLibre()
        registro.puntero <- nuevaPos
        // Escribir nuevo registro en nuevaPos
    FinSi
    Cerrar archivo
FinProceso
```

Elaboración propia.

ISBN: 978-631-6960-03-0

CAPÍTULO 11

EJERCICIOS PROPUESTOS

11.1 TEMA 1: Bucles y tomas de decisión

1. Hacer un pseudocódigo que imprima los números del 1 al 100.
2. Hacer un pseudocódigo que imprima los números del 100 al 0, en orden decreciente.
3. Hacer un pseudocódigo que imprima los números pares entre 0 y 100.
4. Hacer un programa que imprima la suma de los 100 primeros números.
5. Hacer un pseudocódigo que imprima los números impares hasta el 100 y que imprima cuántos impares hay.
6. Hacer un pseudocódigo que imprima todos los números naturales que hay desde la unidad hasta un número que introducimos por teclado.
7. Introducir tantas frases como queramos y contarlas.
8. Hacer un pseudocódigo que solo nos permita introducir S o N.
9. Introducir un número por teclado. Que nos diga si es positivo o negativo.
10. Introducir un número por teclado. Que nos diga si es par o impar.
11. Imprimir y contar los múltiplos de 3 desde la unidad hasta un número que introducimos por teclado.
12. Hacer un pseudocódigo que imprima los números del 1 al 100. Que calcule la suma de todos los números pares por un lado, y por otro, la de todos los impares.
13. Imprimir y contar los números que son múltiplos de 2 o de 3 que hay entre 1 y 100.
14. Hacer un pseudocódigo que imprima el mayor y el menor de una serie de cinco números que vamos introduciendo por teclado.

ISBN: 978-631-6960-03-0

15. Introducir dos números por teclado. Imprimir los números naturales que hay entre ambos números empezando por el más pequeño, contar cuántos hay y cuántos de ellos son pares. Calcular la suma de los impares.

11.2 TEMA 2: Bucles anidados y subprogramas

16. Imprimir diez veces la serie de números del 1 al 10.

17. Imprimir, contar y sumar los múltiplos de 2 que hay entre una serie de números, tal que el segundo sea mayor o igual que el primero.

18. Hacer un pseudocódigo que cuente las veces que aparece una determinada letra en una frase que introduciremos por teclado.

19. Hacer un pseudocódigo que simule el funcionamiento de un reloj digital y que permita ponerlo en hora.

20. Calcular el factorial de un número, mediante subprogramas.

21. Hacer un programa que calcule independientemente la suma de los pares y los impares de los números entre 1 y 1000, utilizando un switch.

11.3 TEMA 3: Presentación en pantalla y cabeceras

22. Introducir una frase por teclado. Imprimirla cinco veces en filas consecutivas, pero cada impresión irá desplazada cuatro columnas hacia la derecha.

23. Hacer un pseudocódigo que imprima los números del 0 al 100, controlando las filas y las columnas.

24. Comprobar si un número mayor o igual que la unidad es primo.

25. Introducir un número menor de 5000 y pasarlo a número romano.

26. Introducir una frase por teclado. Imprimirla en el centro de la pantalla.

27. Realizar la tabla de multiplicar de un número entre 0 y 10.

ISBN: 978-631-6960-03-0

11.4 TEMA 4: Números aleatorios y menús

28. Simular el lanzamiento de una moneda al aire e imprimir si ha salido cara o cruz.
29. Simular cien tiradas de dos dados y contar las veces que entre los dos suman 10.
30. Simular una carrera de dos caballos si cada uno tiene igual probabilidad de ganar.
31. Introducir dos números por teclado y mediante un menú, calcule su suma, su resta, su multiplicación o su división.
32. Hacer un programa que nos permita introducir un número por teclado y sobre él se realicen las siguientes operaciones: comprobar si es primo, hallar su factorial o imprimir su tabla de multiplicar.

11.5 TEMA 5: Arrays unidimensionales

33. Crear un array unidimensional de 20 elementos con nombres de personas. Visualizar los elementos de la lista debiendo ir cada uno en una fila distinta.
34. Hacer un programa que lea las calificaciones de un alumno en 10 asignaturas, las almacene en un vector y calcule e imprima su media.
35. Usando el segundo ejemplo, hacer un programa que busque una nota en el vector.

11.6 TEMA 6: Arrays bidimensionales

36. Generar una matriz de 4 filas y 5 columnas con números aleatorios entre 1 y 100, e imprimirla.
37. Generar una matriz de 4 filas y 5 columnas con números aleatorios entre 1 y 100, y hacer su matriz transpuesta.
38. Cargar en una matriz las notas de los alumnos de un colegio en función del número de cursos (filas) y del número de alumnos por curso (columnas).
39. Ordenar una matriz de M filas y N columnas por la primera columna utilizando el método SHELL (por inserción).

ISBN: 978-631-6960-03-0

11.7 TEMA 7: Arrays multidimensionales

40. Crear una tabla de 3 páginas, 4 filas y 5 columnas donde el primer elemento valga 1, el segundo 2, el tercero 3 y así sucesivamente, e imprimirla.

41. Se dispone de una tabla de 5 páginas, 10 filas y 20 columnas, que se refieren al centro, al curso y al número de alumnos de un colegio respectivamente. Imprimir la nota media por curso y la nota media máxima y su centro de pertenencia.

42. Una empresa guarda en una tabla de 3x12x4 las ventas realizadas por sus tres representantes a lo largo de doce meses de sus cuatro productos, VENTAS[representante, mes, producto]. Queremos proyectar el array tridimensional sobre uno de dos dimensiones que represente el total de ventas, TOTAL[mes, producto], para lo cual sumamos las ventas de cada producto de cada mes de todos los representantes. Imprimir ambos arrays.

11.8 TEMA 8: Ficheros

43. Hacer un programa que nos permita dar altas en el fichero secuencial DATOS.DAT, cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA.

44. Hacer un programa que nos permita dar bajas en el fichero DATOS.DAT.

45. Dado el fichero secuencial DATOS.DAT, realizar un programa que nos permita realizar modificaciones cuantas veces deseemos.

46. Tenemos el fichero secuencial DATOS.DAT cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA. Listar por impresora todos los registros cuya provincia sea una determinada que introduciremos por teclado.

47. En el fichero secuencial VENTAS.DAT, están almacenadas las ventas de los productos durante el día, cuyos campos son: NART y VENTAS. Se desea hacer un programa que liste por impresora todas las ventas realizadas durante el día.

48. Dado el fichero secuencial ARTICULOS.DAT, cuyos campos son: NART, ARTÍCULO, PVP, STOCK y MÍNIMO. En otro fichero VENTAS.DAT, están almacenadas las modificaciones de los productos durante el día, cuyos campos son:

ISBN: 978-631-6960-03-0

NART2, VENTAS y TIPO. El campo TIPO puede tomar los valores 0 (venta) y 1 (compra). Se desea hacer un programa que realice una actualización del fichero de ARTICULOS y un listado por impresora de las entradas y salidas de los artículos.

11.9 TEMA 9: Organización aleatoria y secuencial

49. Hacer un pseudocódigo que nos permita dar altas en el fichero DATOS.DAT de organización directa, controlando las altas duplicadas. Los campos son: DNI, NOMBRE, APELLIDOS Y PUNTERO para ambos archivos.

50. Tenemos el fichero DATOS.DAT, que está indexado por el campo APELLIDOS, cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA. Hacer un programa que nos permita listar por pantalla todos los registros del fichero, controlando el salto de página cuando llegue a la línea veinte.

51. Tenemos el fichero DATOS.DAT con la misma estructura anterior, que está indexado por el campo DNI. Crear un programa que nos permita consultar un registro siempre que queramos.

ISBN: 978-631-6960-03-0

CAPÍTULO 12

EJERCICIOS RESUELTOS

12.1 TEMA 1: Bucles y tomas de decisión

1. **Hacer un pseudocódigo que imprima los números del 1 al 100.**

   PROGRAMA contador1
   ENTORNO:
   c <- 0
   ALGORITMO:
   Borrar_pantalla( )
   MIENTRAS c < 101 HACER
   ESCRIBIR c
   c <- c + 1
   FINMIENTRAS
   FINPROGRAMA

2. **Hacer un pseudocódigo que imprima los números del 100 al 0, en orden decreciente.**

   PROGRAMA contador2
   ENTORNO:
   c <- 100
   ALGORITMO:
   Borrar_pantalla( )
   MIENTRAS c <= 0 HACER
   ESCRIBIR c
   c <- c - 1
   FINMIENTRAS
   FINPROGRAMA

3. **Hacer un pseudocódigo que imprima los números pares entre 0 y 100.**

ISBN: 978-631-6960-03-0

```
PROGRAMA pares
ENTORNO:
c <- 2
ALGORITMO:
Borrar_pantalla( )
MIENTRAS c < 101 HACER
ESCRIBIR c
c <- c + 2
FINMIENTRAS
FINPROGRAMA
```

4. **Hacer un programa que imprima la suma de los 100 primeros números.**

```
PROGRAMA suma
ENTORNO:
c <- 1
suma <- 0
ALGORITMO:
Borrar_pantalla( )
MIENTRAS c <= 100 HACER
suma <- suma + c
c <- c + 1
FINMIENTRAS
ESCRIBIR "La suma de los 100 primeros n£meros es: "
ESCRIBIR suma
FINPROGRAMA
```

5. **Hacer un pseudocódigo que imprima los números impares hasta el 100 y que imprima cuántos impares hay.**

```
PROGRAMA impares
ENTORNO:
c <- 1
son <- 0
ALGORITMO:
```

ISBN: 978-631-6960-03-0

```
Borrar_pantalla( )
MIENTRAS c < 100
ESCRIBIR c
c <- c + 2
son <- son + 1
FINMIENTRAS
ESCRIBIR .El n£mero de impares: "
ESCRIBIR son
FINPROGRAMA
```

6. **Hacer un pseudocódigo que imprima todos los números naturales que hay desde la unidad hasta un número que introducimos por teclado.**

```
PROGRAMA natural
ENTORNO:
i <- 0
n <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR Ïntroduce un n£mero: "
LEER n
MIENTRAS i < n HACER
i <- i + 1
ESCRIBIR i
FINMIENTRAS
FINPROGRAMA
```

7. **Introducir tantas frases como queramos y contarlas.**

```
PROGRAMA frases
ENTORNO:
res <- "S"
frase <- Espacios( 30 )
c <- 0
ALGORITMO:
```

ISBN: 978-631-6960-03-0

```
Borrar_pantalla( )
MIENTRAS res = "S"HACER
ESCRIBIR "Frase: "
LEER frase
c <- c + 1
ESCRIBIR "Deseas introducir m s frases (S/N): "
LEER res
FINMIENTRAS
ESCRIBIR .El n£mero de frases introducidas son: "
ESCRIBIR c
FINPROGRAMA
```

8. **Hacer un pseudocódigo que solo nos permita introducir S o N.**

```
PROGRAMA sn
ENTORNO:
res <-
ALGORITMO:
Borrar_pantalla( )
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR Ïntroduce S o N"
LEER res
res <- Convertir_may£sculas( res )
FINMIENTRAS
FINPROGRAMA
```

9. **Introducir un número por teclado. Que nos diga si es positivo o negativo.**

```
PROGRAMA signo
ENTORNO:
num <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR Ïntroduce un n£mero: "
LEER num
```

ISBN: 978-631-6960-03-0

```
SI num >= 0 ENTONCES
ESCRIBIR .es positivo"
SINO
ESCRIBIR .es negativo"
FINSI
FINPROGRAMA
```

10. **Introducir un número por teclado. Que nos diga si es par o impar.**

```
PROGRAMA paridad
ENTORNO:
num <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR Ïntroduce un n£mero: "
LEER num
SI num = int( num / 2 ) * 2 ENTONCES
ESCRIBIR .es par"
SINO
ESCRIBIR .es impar"
FINSI
FINPROGRAMA
```

11. **Imprimir y contar los múltiplos de 3 desde la unidad hasta un número que introducimos por teclado.**

```
PROGRAMA multiplo3
ENTORNO:
i <- 3
n <- 0
c <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR "N£mero: "
LEER n
```

ISBN: 978-631-6960-03-0

```
MIENTRAS i <= n HACER
SI i = int( i / 3 ) * 3 ENTONCES
ESCRIBIR i
c <- c + 1
FINSI
i <- i + 1
FINMIENTRAS
ESCRIBIR .El n£mero de m£ltiplos de 3 son: "
ESCRIBIR c
FINPROGRAMA
```

12. **Hacer un pseudocódigo que imprima los números del 1 al 100. Que calcule la suma de todos los números pares por un lado, y por otro, la de todos los impares.**

```
PROGRAMA par_impar
ENTORNO:
i <- 1
sumapar <- 0
sumaimp <- 0
ALGORITMO:
Borrar_pantalla( )
MIENTRAS i < 101 HACER
SI i = int( i / 2 ) * 2 ENTONCES
sumapar <- sumapar + i
SINO
sumaimp <- sumaimp + i
FINSI
i <- i + 1
FINMIENTRAS
ESCRIBIR "La suma de los pares es: "
ESCRIBIR sumapar
ESCRIBIR "La suma de los impares es: "
ESCRIBIR sumaimp
FINPROGRAMA
```

ISBN: 978-631-6960-03-0

13. **Imprimir y contar los números que son múltiplos de 2 o de 3 que hay entre 1 y 100.**

```
PROGRAMA multiplo_2_3
ENTORNO:
i <- 1
c <- 0
ALGORITMO:
Borrar_pantalla( )
MIENTRAS i < 101 HACER
SI i = int( i / 2 ) * 2 O i = int( i / 3 ) * 3 ENTONCES
c <- c + 1
ESCRIBIR i
FINSI
i <- i + 1
FINMIENTRAS
ESCRIBIR .El n£mero de m£ltiplos es de: "
ESCRIBIR c
FINPROGRAMA
```

14. **Hacer un pseudocódigo que imprima el mayor y el menor de una serie de cinco números que vamos introduciendo por teclado.**

```
PROGRAMA mayor_menor
ENTORNO:
con <- 0
n <- 0
maximo <- 0
minimo <- 99999
ALGORITMO:
Borrar_pantalla( )
MIENTRAS con <= 5 HACER
ESCRIBIR "N£mero: "
LEER n
SI n > maximo ENTONCES
```

ISBN: 978-631-6960-03-0

```
maximo = n
FINSI
SI n < minimo ENTONCES
minimo <- n
FINSI
con <- con + 1
FINMIENTRAS
ESCRIBIR .El mayor de los n£meros es: "
ESCRIBIR maximo
ESCRIBIR .El menor de los n£meros es: "
ESCRIBIR minimo
FINPROGRAMA
```

15. **Introducir dos números por teclado. Imprimir los números naturales que hay entre ambos números empezando por el más pequeño, contar cuántos hay y cuántos de ellos son pares. Calcular la suma de los impares.**

```
PROGRAMA par_impar
ENTORNO:
num1 <- 0
num2 <- 0
aux <- 0
son <- 0
pares <- 0
sumaimpa <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR "N£mero: "
LEER num1
ESCRIBIR "N£mero: "
LEER num2
SI num1 > num2 ENTONCES
aux <- num1
num1 <- num2
```

ISBN: 978-631-6960-03-0

```
num2 <- aux
FINSI
MIENTRAS num1 >= num2 HACER
ESCRIBIR num1
son <- son + 1
SI num1 = int( num1 / 2 ) * 2 ENTONCES
pares <- pares + 1
SINO
sumaimpa <- sumaimpa + num1
FINSI
num1 <- num1 + 1
FINMIENTRAS
ESCRIBIR "N£meros visualizados: "
ESCRIBIR son
ESCRIBIR "Pares hay: "
ESCRIBIR pares
ESCRIBIR "La suma de los impares es: "
ESCRIBIR sumaimpa
FINPROGRAMA
```

12.2 TEMA 2: Bucles anidados y subprogramas

16. **Imprimir diez veces la serie de números del 1 al 10.**

```
PROGRAMA diez
ENTORNO:
serie <- 0
ALGORITMO:
Borrar_pantalla( )
MIENTRAS serie <= 10 HACER
numero <- 1
MIENTRAS numero <= 10 HACER
ESCRIBIR numero
numero <- numero + 1
```

ISBN: 978-631-6960-03-0

```
FINMIENTRAS
serie <- serie + 1
FINMIENTRAS
FINPROGRAMA
```

17. **Imprimir, contar y sumar los múltiplos de 2 que hay entre una serie de números, tal que el segundo sea mayor o igual que el primero.**

```
PROGRAMA multiplo2
ENTORNO:
res <- "S"
ALGORITMO:
Borrar_pantalla( )
MIENTRAS res = "S"HACER
c <- 0
sum <- 0
num1 <- 0
num2 <- -999
ESCRIBIR "N£mero: "
LEER num1
ESCRIBIR "N£mero mayor que el anterior"
MIENTRAS num1 >= num2 HACER
LEER num2
FINMIENTRAS
num1 <- num1 + 1
MIENTRAS num1 <= num2 - 1 HACER
SI num1 = int( num1 / 2 ) * 2 ENTONCES
ESCRIBIR num1
c <- c + 1
sum <- sum + num1
FINSI
num1 <- num1 + 1
FINMIENTRAS
ESCRIBIR "N£mero de m£ltiplos de 2: "
```

ISBN: 978-631-6960-03-0

```
ESCRIBIR c
ESCRIBIR "Su suma es: "
ESCRIBIR sum
res <- Espacios( 1 )
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR .ºtra serie de n£meros (S/N): "
LEER res
res <- Convertir_may£sculas( res )
FINMIENTRAS
FINMIENTRAS
FINPROGRAMA
```

18. **Hacer un pseudocódigo que cuente las veces que aparece una determinada letra en una frase que introduciremos por teclado.**

```
PROGRAMA letra
ENTORNO:
frase <- Espacios( 30 )
letra <- Espacios( 1 )
longitud <- 0
a <- 0
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
ESCRIBIR Ïntroduce una frase: "
LEER frase
longitud <- Hallar_longitud( frase )
i <- 1
ESCRIBIR "Letra a buscar: "
LEER letra
MIENTRAS i <= longitud HACER
SI letra = Caracter( frase, i, 1 ) ENTONCES
a <- a + 1
```

ISBN: 978-631-6960-03-0

```
FINSI
i <- i + 1
FINMIENTRAS
Borrar_pantalla( )
ESCRIBIR .El n£mero de veces que aparece la letra "
ESCRIBIR letra
ESCRIBIR .en la frase "
ESCRIBIR frase
ESCRIBIR .es de "
ESCRIBIR a
res <- Espacios( 1 )
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea introducir m&aacute;s frases (S/N): "
LEER res
res <- Convertir_may£sculas( res )
FINMIENTRAS
FINMIENTRAS
FINPROGRAMA
```

19. **Hacer un pseudocódigo que simule el funcionamiento de un reloj digital y que permita ponerlo en hora.**

```
PROGRAMA reloj
ENTORNO:
horas <- 0
minutos <- 0
segundos <- 0
res <- "S"
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR "Horas: "
LEER horas
ESCRIBIR "Minutos: "
LEER minutos
```

ISBN: 978-631-6960-03-0

```
ESCRIBIR "Segundos: "
LEER segundos
MIENTRAS res = "S"HACER
MIENTRAS horas < 24 HACER
MIENTRAS minutos < 60 HACER
MIENTRAS segundos < 60 HACER
ESCRIBIR horas
ESCRIBIR minutos
ESCRIBIR segundos
segundos <- segundos + 1
FINMIENTRAS
minutos <- minutos + 1
segundos <- 0
FINMIENTRAS
horas <- horas + 1
minutos <- 0
FINMIENTRAS
horas <- 0
FINMIENTRAS
FINPROGRAMA
```

20. **Calcular el factorial de un número, mediante subprogramas.**

```
PROGRAMA factorial
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
factorial <- 1
ESCRIBIR "N£mero: "
LEER numero
SI numero < 0 ENTONCES
ESCRIBIR "No tiene factorial"
```

ISBN: 978-631-6960-03-0

SINO
HACER Calculos
FINSI
HACER Mas
FINMIENTRAS
FINPROGRAMA

SUBPROGRAMA Calculos
MIENTRAS numero > 1 HACER
*factorial <- factorial * numero*
numero <- numero - 1
FINMIENTRAS
HACER Imprimir
FINSUBPROGRAMA

SUBPROGRAMA Mas
res <-
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea calcular más factoriales (S/N): "
LEER res
res <- Convertir_may£sculas(res)
FINMIENTRAS
FINSUBPROGRAMA

SUBPROGRAMA Imprimir
ESCRIBIR "Su factorial es: "
ESCRIBIR factorial
FINSUBPROGRAMA

21. **Hacer un programa que calcule independientemente la suma de los pares y los impares de los números entre 1 y 1000, utilizando un switch.**

PROGRAMA suma
ENTORNO:

ISBN: 978-631-6960-03-0

```
par <- 0
impar <- 0
sw <- 0
i <- 1
ALGORITMO:
Borrar_pantalla( )
MIENTRAS i <= 1000 HACER
SI sw = 0 ENTONCES
impar <- impar + i
sw <- 1
SINO
par <- par + i
sw <- 0
FINSI
i <- i + 1
FINMIENTRAS
ESCRIBIR "La suma de los pares es: "
ESCRIBIR par
ESCRIBIR "La suma de los impares es: "
ESCRIBIR impar
FINPROGRAMA
```

12.3 TEMA 3: Presentación en pantalla y cabeceras

22. **Introducir una frase por teclado. Imprimirla cinco veces en filas consecutivas, pero cada impresión irá desplazada cuatro columnas hacia la derecha.**

```
PROGRAMA frase
ENTORNO:
frase <- Espacios( 30 )
ALGORITMO:
Borrar_pantalla( )
EN 5,15 ESCRIBIR "Frase: "
EN 5,22 LEER frase
```

ISBN: 978-631-6960-03-0

```
fi <- 8
co <- 15
veces <- 0
MIENTRAS veces <= 5 HACER
EN fi,co ESCRIBIR frase
veces <- veces + 1
co <- co + 4
fi <- fi + 1
FINMIENTRAS
FINPROGRAMA
```

23. **Hacer un pseudocódigo que imprima los números del 0 al 100, controlando las filas y las columnas.**

```
PROGRAMA numeros
ENTORNO:
c <- 0
ALGORITMO:
Borrar_pantalla( )
EN 5,20 ESCRIBIR "Los n£meros del 0 al 100 son: "
fi <- 7
col <- 5
MIENTRAS c < 101 HACER
EN fi,col ESCRIBIR c
c <- c + 1
col <- col + 4
SI col > 75 ENTONCES
fi <- fi + 2
col <- 5
FINSI
FINMIENTRAS
FINPROGRAMA
```

24. **Comprobar si un número mayor o igual que la unidad es primo.**

ISBN: 978-631-6960-03-0

```
PROGRAMA primo
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
numero <- 0
sw <- 0
MIENTRAS numero < 1 HACER
EN 8,10 ESCRIBIR "N£mero: "
EN 8,18 LEER numero
FINMIENTRAS
i <- numero - 1
MIENTRAS i > 1 Y sw <> 1 HACER
SI numero = Int( numero / i ) * i ENTONCES
sw = 1
SINO
i <- i - 1
FINSI
FINMIENTRAS
SI sw = 1 ENTONCES
EN 10,10 ESCRIBIR "no es primo"
SINO
EN 10,10 ESCRIBIR "s¡ es primo"
FINSI
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
res <-
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea introducir m s n£meros (S/N): "
```

ISBN: 978-631-6960-03-0

```
LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
```

25. **Introducir un número menor de 5000 y pasarlo a número romano.**

```
PROGRAMA romano
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
num <- 0
MIENTRAS num < 1 O num > 5000 HACER
EN 8,10 ESCRIBIR "N£mero: "
EN 8,18 ESCRIBIR num
FINMIENTRAS
col <- 15
MIENTRAS num >= 1000 HACER
EN 15,col ESCRIBIR "M"
num <- num - 1000
col <- col + 1
FINMIENTRAS
SI num >= 900 ENTONCES
EN 15,col ESCRIBIR ÇM"
num <- num - 900
col <- col + 2
FINSI
SI num >= 500 ENTONCES
EN 15,col ESCRIBIR "D"
num <- num - 500
col <- col + 1
FINSI
```

ISBN: 978-631-6960-03-0

```
MIENTRAS num >= 100 HACER
EN 15,col ESCRIBIR Ç"
num <- num - 100
col <- col + 1
FINMIENTRAS
SI num >= 90 ENTONCES
EN 15,col ESCRIBIR "XC"
num <- num - 90
col <- col + 2
FINSI
SI num >= 50 ENTONCES
EN 15,col ESCRIBIR "L"
num <- num - 50
col <- col + 1
FINSI
SI num >= 40 ENTONCES
EN 15,col ESCRIBIR "XL"
num <- num - 40
col <- col + 2
FINSI
MIENTRAS num >= 10 HACER
EN 15,col ESCRIBIR "X"
num <- num - 10
col <- col + 1
FINMIENTRAS
SI num = 9 ENTONCES
EN 15,col ESCRIBIR ÏX"
num <- num - 9
col <- col + 2
FINSI
SI num >= 5 ENTONCES
EN 15,col ESCRIBIR "V"
num <- num - 5
```

ISBN: 978-631-6960-03-0

```
col <- col + 1
FINSI
SI num >= 4 ENTONCES
EN 15,col ESCRIBIR ÏV"
num <- num - 4
col <- col + 2
FINSI
MIENTRAS num > 0 HACER
EN 15,col ESCRIBIR Ï"
num <- num - 1
col <- col + 1
FINMIENTRAS
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
res <-
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea introducir m s n£meros (S/N): "
LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
```

26. **Introducir una frase por teclado. Imprimirla en el centro de la pantalla.**

```
PROGRAMA centro
ENTORNO:
res <- "S"
frase <- Espacios( 40 )
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
```

ISBN: 978-631-6960-03-0

```
EN 5,15 ESCRIBIR "Frase: "
EN 5,22 LEER frase
EN 12,40 - Int( Longitud( frase ) / 2 ) ESCRIBIR frase
HACER Mas
FINMIENTRAS
FINPROGRAMA
```

27. **Realizar la tabla de multiplicar de un número entre 0 y 10.**

```
PROGRAMA tabla
ENTORNO:
num <- -1
ALGORITMO:
HACER Numero
Borrar_pantalla( )
EN 5,10 ESCRIBIR "Tabla de multiplicar del n£mero: "
EN 5,40 LEER num
i <- 0
fi <- 8
MIENTRAS i <= 10 HACER
EN fi,15 ESCRIBIR num
EN fi,19 ESCRIBIR "* "
EN fi,23 ESCRIBIR i
EN fi,25 ESCRIBIR - "
EN fi,29 ESCRIBIR num * i
fi <- fi + 1
i <- i + 1
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Numero
MIENTRAS num < 0 HACER
Borrar_pantalla( )
EN 10,25 ESCRIBIR "N£mero: "
```

ISBN: 978-631-6960-03-0

```
EN 10,33 LEER num
FINMIENTRAS
FINSUBPROGRAMA
```

12.4 TEMA 4: Números aleatorios y menús

28. **Simular el lanzamiento de una moneda al aire e imprimir si ha salido cara o cruz.**

```
PROGRAMA moneda
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
Borrar_pantalla( )
SI Rnd( ) <= 0.5 ENTONCES
EN 10,35 ESCRIBIR Çara"
SINO
EN 10,35 ESCRIBIR Çruz"
FINSI
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
res <- Espacios( 1 )
MIENTRAS res <> "S"Y res <> "N"HACER
EN 20,30 ESCRIBIR "M s lanzamientos (S/N): "
EN 20,57 LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
```

29. **Simular cien tiradas de dos dados y contar las veces que entre los dos suman 10.**

ISBN: 978-631-6960-03-0

```
PROGRAMA dado
ENTORNO:
c <- 0
i <- 0
ALGORITMO:
Borrar_pantalla( )
MIENTRAS i < 101 HACER
SI Int( Rnd( ) * 6 ) + Int( Rnd( ) * 6 ) + 2 = 10 ENTONCES
c <- c + 1
FINSI
i <- i + 1
FINMIENTRAS
EN 10,20 ESCRIBIR "Las veces que suman 10 son: "
EN 10,48 ESCRIBIR c
FINPROGRAMA
```

30. **Simular una carrera de dos caballos si cada uno tiene igual probabilidad de ganar.**

```
PROGRAMA caballos
ENTORNO:
dibujo <- "****"
col1 <- 4
col2 <- 4
ALGORITMO:
Borrar_pantalla( )
EN 10,col1 ESCRIBIR dibujo
EN 10,col2 ESCRIBIR dibujo
MIENTRAS col1 <= 75 Y col2 <= 75 HACER
SI Rnd( ) <= 0.5 ENTONCES
EN 10,col1 ESCRIBIR Espacios( 4 )
col1 <- col1 + 4
EN 10,col1 ESCRIBIR dibujo
SINO
```

ISBN: 978-631-6960-03-0

```
EN 12,col2 ESCRIBIR Espacios( 4 )
col2 <- col2 + 4
EN 12,col2 ESCRIBIR dibujo
FINSI
FINMIENTRAS
EN 16,20 ESCRIBIR .El ganador es el caballo n£mero: "
SI col1 >= 75 ENTONCES
EN 16,54 ESCRIBIR "1"
SINO
EN 16,54 ESCRIBIR "2"
FINSI
FINPROGRAMA
```

31. **Introducir dos números por teclado y mediante un menú, calcule su suma, su resta, su multiplicación o su división.**

```
PROGRAMA menu1
ENTORNO:
op <- 0
ALGORITMO:
EN 10,20 ESCRIBIR "N£mero: "
EN 10,29 LEER n1
EN 12,20 ESCRIBIR "N£mero: "
EN 12,29 LEER n2
MIENTRAS op <> 5 HACER
op <- 0
Borrar_pantalla( )
EN 6,20 ESCRIBIR "Men£ de opciones"
EN 10,25 ESCRIBIR "1.- Suma"
EN 12,25 ESCRIBIR "2.- Resta"
EN 14,25 ESCRIBIR "3.- Multiplicaci¢n"
EN 16,25 ESCRIBIR "4.- Divisi¢n"
EN 18,25 ESCRIBIR "5.- Salir del programa"
EN 22,25 ESCRIBIR .Elija opci¢n: "
```

ISBN: 978-631-6960-03-0

```
EN 22,39 LEER op
Borrar_pantalla( )
HACER CASO
CASO op = 1
EN 10,20 ESCRIBIR "Su suma es: "
EN 10,33 ESCRIBIR n1 + n2
Pausa( )
CASO op = 2
EN 10,20 ESCRIBIR "Su resta es: "
EN 10,33 ESCRIBIR n1 - n2
Pausa( )
CASO op = 3
EN 10,20 ESCRIBIR "Su multiplicaci¢n es: "
EN 10,33 ESCRIBIR n1 * n2
Pausa( )
CASO op = 4
EN 10,20 ESCRIBIR "Su divisi¢n es: "
EN 10,33 ESCRIBIR n1 / n2
Pausa( )
FINCASO
FINMIENTRAS
FINPROGRAMA
```

32. **Hacer un programa que nos permita introducir un número por teclado y sobre él se realicen las siguientes operaciones: comprobar si es primo, hallar su factorial o imprimir su tabla de multiplicar.**

```
PROGRAMA menu2
ENTORNO:
op <- 0
ALGORITMO:
EN 10,20 ESCRIBIR "N£mero: "
EN 10,29 LEER n
MIENTRAS op <> 4 HACER
```

ISBN: 978-631-6960-03-0

```
op <- 0
Borrar_pantalla( )
EN 6,30 ESCRIBIR "Men£ de opciones"
EN 10,25 ESCRIBIR "1.- Comprobar si es primo"
EN 12,25 ESCRIBIR "2.- Hallar su factorial"
EN 14,25 ESCRIBIR "3.- Tabla de multiplicar"
EN 16,25 ESCRIBIR "4.- Salir del programa"
EN 22,25 ESCRIBIR .Elija opci¢n: "
EN 22,39 LEER op
HACER CASO
CASO op = 1
HACER Primo
CASO op = 2
HACER Factorial
CASO op = 3
HACER Tabla
FINCASO
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Primo
sw <- 0
i <- n - 1
MIENTRAS i > 1 Y sw <> 1 HACER
SI n = Int( n / i ) * i ENTONCES
sw <- 1
SINO
i <- i - 1
FINSI
FINMIENTRAS
Borrar_pantalla( )
SI sw = 1 ENTONCES
EN 10,10 ESCRIBIR "no es primo"
```

ISBN: 978-631-6960-03-0

```
SINO
EN 10,10 ESCRIBIR "s¡ es primo"
FINSI
Pausa( )
FINSUBPROGRAMA
-----------------------------
SUBPROGRAMA Factorial
fac <- 1
Borrar_pantalla( )
SI n < 0 ENTONCES
EN 10,10 ESCRIBIR "No tiene factorial"
SINO
MIENTRAS n > 1 HACER
fac <- fac * n
n <- n - 1
FINMIENTRAS
EN 10,10 ESCRIBIR "Su factorial es: "
EN 10,27 ESCRIBIR fac
FINSI
Pausa( )
FINSUBPROGRAMA
-----------------------------
SUBPROGRAMA Tabla
i <- 0
fi <- 10
Borrar_pantalla( )
MIENTRAS i <= 10 HACER
EN 8,10 ESCRIBIR "Tabla de multiplicar"
EN fi,10 ESCRIBIR n
EN fi,15 ESCRIBIR "*"
EN fi,20 ESCRIBIR i
EN fi,25 ESCRIBIR -"
EN fi,30 ESCRIBIR n * i
```

ISBN: 978-631-6960-03-0

```
i <- i + 1
FINMIENTRAS
Pausa( )
FINSUBPROGRAMA
```

12.5 TEMA 5: Arrays unidimensionales

33. **Crear un array unidimensional de 20 elementos con nombres de personas. Visualizar los elementos de la lista debiendo ir cada uno en una fila distinta.**

```
PROGRAMA nombres
ENTORNO:
DIMENSIONA datos[ 20 ]
i <- 1
ALGORITMO:
Borrar_pantalla( )
fi <- 10
MIENTRAS i < 21 HACER
EN fi,10 ESCRIBIR "Nombre: "
EN fi, 18 LEER datos[ i ]
i <- i + 1
FINMIENTRAS
Borrar_pantalla( )
i <- 1
fi <- 3
EN 1,20 ESCRIBIR .Elementos de la lista"
MIENTRAS i < 21 HACER
EN fi,28 ESCRIBIR datos[ i ]
fi <- fi + 1
i <- i + 1
FINMIENTRAS
FINPROGRAMA
```

34. **Hacer un programa que lea las calificaciones de un alumno en 10 asignaturas, las almacene en un vector y calcule e imprima su media.**

ISBN: 978-631-6960-03-0

```
PROGRAMA notamedia
ENTORNO:
DIMENSIONA notas[ 10 ]
suma <- 0
media <- 0
ALGORITMO:
Borrar_pantalla( )
fi <- 7
PARA i DESDE 1 HASTA 10 HACER
EN fi,15 ESCRIBIR "Nota "
EN fi,20 ESCRIBIR i
EN fi,21 ESCRIBIR ": "
EN fi,23 LEER notas[ i ]
fi <- fi + 1
FINPARA
PARA i DESDE 1 HASTA 10 HACER
suma <- suma + notas[ i ]
FINPARA
media <- suma / 10
EN 20,20 ESCRIBIR "Nota media: "
EN 20,32 ESCRIBIR media
FINPROGRAMA
```

35. **Usando el segundo ejemplo, hacer un programa que busque una nota en el vector.**

```
PROGRAMA buscar
ENTORNO:
i <- 0
num <- 0
ALGORITMO:
Borrar_pantalla( )
ESCRIBIR "Nota a buscar: "
LEER num
```

ISBN: 978-631-6960-03-0

```
ITERAR
i <- i + 1
SI notas[ i ] = num O i = 10 ENTONCES
SALIR
FINSI
FINITERAR
SI notas[ i ] = num ENTONCES
ESCRIBIR .En contrado en posici&oacute;n: "
ESCRIBIR i
SINO
ESCRIBIR "No existe esa nota"
FINSI
FINPROGRAMA
```

12.6 TEMA 6: Arrays bidimensionales

36. **Generar una matriz de 4 filas y 5 columnas con números aleatorios entre 1 y 100, e imprimirla.**

```
PROGRAMA matriz
ENTORNO:
DIMENSIONAR A[ 4, 5 ]
i <- 1
fi <- 10
co <- 15
ALGORITMO:
Borrar_pantalla( )
EN 6,25 ESCRIBIR .El ementos de la matriz"
MIENTRAS i <= 4 HACER
j <- 1
MIENTRAS j <= 5 HACER
A[ i, j ] <- Int( Rnd( ) * 100 ) + 1
EN fi,co ESCRIBIR A[ i, j ]
co <- co + 5
```

ISBN: 978-631-6960-03-0

```
j <- j + 1
FINMIENTRAS
co <- 15
fi <- fi + 2
i <- i + 1
FINMIENTRAS
FINPROGRAMA
```

37. **Generar una matriz de 4 filas y 5 columnas con números aleatorios entre 1 y 100, y hacer su matriz transpuesta.**

```
PROGRAMA transpuesta
ENTORNO:
DIMENSIONAR A[ 4, 5 ]
DIMENSIONAR B[ 5, 4 ]
fi <- 8
co <- 10
fit <- 8
cot <- 40
i <- 1
ALGORITMO:
Borrar_pantalla( )
EN 6,15 ESCRIBIR "Matriz uno"
EN 6,45 ESCRIBIR "Transpuesta"
MIENTRAS i <= 4 HACER
j <- 1
MIENTRAS j <= 5 HACER
A[ i, j ] <- Int( Rnd( ) * 100 ) + 1
B[ j, i ] <- A[ i, j ]
EN fi,co ESCRIBIR A[ i, j ]
EN fit,cot ESCRIBIR B[ j, i ]
co <- co + 4
fit <- fit + 2
j <- j + 1
```

ISBN: 978-631-6960-03-0

```
FINMIENTRAS
fi <- fi + 2
co <- 10
fit <- 8
cot <- cot + 4
i <- i + 1
FINMIENTRAS
FINPROGRAMA
```

38. **Cargar en una matriz las notas de los alumnos de un colegio en función del número de cursos (filas) y del número de alumnos por curso (columnas).**

```
PROGRAMA notas
ENTORNO:
i <- 1
j <- 1
ALGORITMO:
Borrar_pantalla( )
EN 10,20 ESCRIBIR "N£mero de cursos: "
EN 10,39 LEER M
EN 12,20 ESCRIBIR "N£mero de alumnos: "
EN 12,40 LEER N
DIMENSIONAR A[ M, N ]
Borrar_pantalla( )
EN 2,25 ESCRIBIR Ïntroducci¢n de las notas"
MIENTRAS i <= M HACER
EN 10,25 ESCRIBIR Çurso: "
EN 10,32 ESCRIBIR i
MIENTRAS j <= N HACER
EN 14,25 ESCRIBIR .ªlumno: "
EN 14,33 ESCRIBIR j
EN 16,25 ESCRIBIR "Nota: "
EN 16,32 LEER A[ i, j ]
j <- j + 1
```

ISBN: 978-631-6960-03-0

```
FINMIENTRAS
i <- i + 1
FINMIENTRAS
FINPROGRAMA
```

39. **Ordenar una matriz de M filas y N columnas por la primera columna utilizando el método SHELL (por inserción).**

```
PROGRAMA ordenar
ENTORNO:
i <- 1
j <- 1
fi <- 10
co <- 15
M <- 0
N <- 0
ALGORITMO:
Borrar_pantalla( )
EN 10,20 ESCRIBIR "Filas: "
EN 10,27 LEER M
EN 12,20 ESCRIBIR Çolumnas: "
EN 12,30 LEER N
DIMENSIONAR A[ M, N ]
Borrar_pantalla( )
MIENTRAS i <= M HACER
MIENTRAS j <= N HACER
A[ i, j ] = Int( Rnd( ) * 100 ) + 1
EN fi,co ESCRIBIR A[ i, j ]
co <- co + 5
j <- j + 1
FINMIENTRAS
co <- 15
fi <- fi + 2
i <- i + 1
```

ISBN: 978-631-6960-03-0

```
FINMIENTRAS
salto <- Int( M / 2 )
MIENTRAS salto >= 1 HACER
sw <- 1
MIENTRAS sw <> 0 HACER
sw <- 0
i <- 1
MIENTRAS i <= M - salto HACER
SI A[ i, 1 ] > A[ i + salto, 1 ] ENTONCES
HACER Cambios
FINSI
i <- i + 1
FINMIENTRAS
FINMIENTRAS
salto <- Int( salto / 2 )
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Cambios
j <- 1
MIENTRAS j <= N HACER
aux <- A[ i + salto, j ]
A[ i + salto, j ] <- A[ i, j ]
A[ i, j ] <- aux
j <- j + 1
FINMIENTRAS
sw <- 1
FINSUBPROGRAMA
```

12.7 TEMA 7: Arrays multidimensionales

40. **Crear una tabla de 3 páginas, 4 filas y 5 columnas donde el primer elemento valga 1, el segundo 2, el tercero 3 y así sucesivamente, e imprimirla.**

ISBN: 978-631-6960-03-0

```
PROGRAMA tabla
ENTORNO:
DIMENSIONAR A[ 3, 4, 5 ]
i <- 1
j <- 1
k <- 1
b <- 0
fi <- 8
co <- 12
ALGORITMO:
MIENTRAS i <= 3 HACER
Borrar_pantalla( )
EN fi,co ESCRIBIR .Elementos de la p gina: "
EN fi,co + 24 ESCRIBIR i
fi <- fi + 2
MIENTRAS j <= 4 HACER
MIENTRAS k <= 5 HACER
b <- b + 1
A[ i, j, k ] <- b
EN fi,co ESCRIBIR A[ i, j, k ]
co <- co + 4
k <- k + 1
FINMIENTRAS
fi <- fi + 2
co <- 12
j <- j + 1
FINMIENTRAS
EN fi + 2,20 ESCRIBIR "Pulse INTRO para continuar ..."
Pausa( )
i <- i + 1
FINMIENTRAS
FINPROGRAMA
```

41. **Se dispone de una tabla de 5 páginas, 10 filas y 20 columnas, que se refieren**

ISBN: 978-631-6960-03-0

al centro, al curso y al número de alumnos de un colegio respectivamente. Imprimir la nota media por curso y la nota media máxima y su centro de pertenencia.

```
PROGRAMA notas
ENTORNO:
max <- -1
sum <- 0
centro <- 0
i <- 1
j <- 1
k <- 1
fi <- 10
ALGORITMO:
Borrar_pantalla( )
EN 8,18 ESCRIBIR Çentro"
EN 8,38 ESCRIBIR "Nota media"
MIENTRAS i <= 5 HACER
MIENTRAS j <= 10 HACER
MIENTRAS k <= 20 HACER
sum <- sum + A[ i, j, k ]
k <- k + 1
FINMIENTRAS
j <- j + 1
FINMIENTRAS
EN fi,20 ESCRIBIR i
EN fi,40 ESCRIBIR sum / 20
fi <- fi + 2
SI sum / 20 > max ENTONCES
max <- sum / 20
centro <- i
FINSI
i <- i + 1
FINMIENTRAS
```

ISBN: 978-631-6960-03-0

EN fi + 2,20 ESCRIBIR "Nota media m xima: "
EN fi + 2,39 ESCRIBIR max
EN fi + 4, 20 ESCRIBIR "pertenece al centro: "
EN fi + 4,41 ESCRIBIR centro
FINPROGRAMA

42. **Una empresa guarda en una tabla de 3x12x4 las ventas realizadas por sus tres representantes a lo largo de doce meses de sus cuatro productos, VENTAS[representante, mes, producto]. Queremos proyectar el array tridimensional sobre uno de dos dimensiones que represente el total de ventas, TOTAL[mes, producto], para lo cual sumamos las ventas de cada producto de cada mes de todos los representantes. Imprimir ambos arrays.**

PROGRAMA ventas
ENTORNO:
*** Las variables est n definidas en cada subprograma*
ALGORITMO:
HACER Volcar
HACER Imp_Tres
HACER Imp_Dos
FINPROGRAMA

SUBPROGRAMA Volcar
DIMENSIONAR TOTAL[12, 4]
j <- 1
MIENTRAS j <= 12 HACER
k <- 1
MIENTRAS k <= 4 HACER
i <- 1
suma <- 0
MIENTRAS i <= 3 HACER
suma <- suma + VENTAS[i, j, k]
i <- i + 1
FINMIENTRAS

ISBN: 978-631-6960-03-0

```
TOTAL[ j, k ] <- suma
k <- k + 1
FINMIENTRAS
j <- j + 1
FINMIENTRAS
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Imp_Tres
i <- 1
MIENTRAS i <= 3 HACER
Borrar_pantalla( )
fi <- 8
co <- 12
EN fi,co ESCRIBIR "Ventas del representante: "
EN fi,co + 26 ESCRIBIR i
fi <- fi + 2
j <- 1
MIENTRAS j <= 12 HACER
k <- 1
MIENTRAS k <= 4 HACER
EN fi,co ESCRIBIR VENTAS[ i, j, k ]
co <- co + 4
k <- k + 1
FINMIENTRAS
fi <- fi + 2
co <- 12
j <- j + 1
FINMIENTRAS
Pausa( )
i <- i + 1
FINMIENTRAS
FINSUBPROGRAMA
------------------------------
```

ISBN: 978-631-6960-03-0

```
SUBPROGRAMA Imp_Dos
Borrar_pantalla( )
j <- 1
EN 8,20 ESCRIBIR "Ventas totales"
fi <- 10
co <- 16
MIENTRAS j <= 12 HACER
k <- 1
MIENTRAS k <= 4 HACER
EN fi,co ESCRIBIR TOTAL[ j, k ]
co <- co + 4
k <- k + 1
FINMIENTRAS
fi <- fi + 2
co <- 12
j <- j + 1
FINMIENTRAS
FINSUBPROGRAMA
```

12.8 TEMA 8: Ficheros

43. **Hacer un programa que nos permita dar altas en el fichero secuencial DATOS.DAT, cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA.**

```
PROGRAMA altas
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
ABRIR "DATOS.DAT"
sw <- 0
num <- 0
Borrar_pantalla( )
```

ISBN: 978-631-6960-03-0

```
EN 5,10 ESCRIBIR "D.N.I.: "
EN 5,18 LEER num
MIENTRAS NO Eof( ) Y sw = 0 HACER
SI dni = num ENTONCES
EN 10,10 ESCRIBIR .Alta duplicada"
EN 15,10 ESCRIBIR "Pulse INTRO para continuar"
Pausa( )
sw <- 1
SINO
Siguiente_registro( )
FINSI
FINMIENTRAS
SI sw = 0 ENTONCES
EN 7,5 ESCRIBIR "Nombre: "
EN 9,5 ESCRIBIR .Apellidos: "
EN 11,5 ESCRIBIR "Direcci&oacute;n: "
EN 13,5 ESCRIBIR "Provincia: "
EN 7,16 LEER nombre
EN 9,16 LEER apellidos
EN 11,16 LEER direccion
EN 13,16 LEER provincia
dni <- num
Final_fichero( )
Escribir_registro( )
FINSI
CERRAR "DATOS.DAT"
res <- Espacios( 1 )
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
MIENTRAS res <> "S"Y res <> "N"HACER
```

ISBN: 978-631-6960-03-0

```
ESCRIBIR "Desea m s altas (S/N): "
LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
```

44. Hacer un programa que nos permita dar bajas en el fichero DATOS.DAT.

```
PROGRAMA bajas
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
ABRIR "DATOS.DAT"
sw <- 0
Borrar_pantalla( )
EN 5,10 ESCRIBIR "D.N.I.: "
EN 5,18 LEER num
MIENTRAS NO Eof( ) HACER
SI dni = num ENTONCES
sw <- 1
SINO
ABRIR .AUX.DAT"
Final_fichero( )
Escribir_registro( )
FINSI
ABRIR "DATOS.DAT"
Siguiente_registro( )
FINMIENTRAS
CERRAR "DATOS.DAT"
CERRAR .AUX.DAT"
SI sw = 0 ENTONCES
EN 12,10 ESCRIBIR "Baja inexistente"
EN 16,10 ESCRIBIR "Pulse INTRO para continuar"
```

ISBN: 978-631-6960-03-0

```
BORRAR .ªUX.DAT"
Pausa( )
SINO
BORRAR "DATOS.DAT"
RENOMBRAR .ªUX.DATÇOMO "DATOS.DAT"
FINSI
res = Espacios( 1 )
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea m s bajas (S/N): "
LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
```

45. **Dado el fichero secuencial DATOS.DAT, realizar un programa que nos permita realizar modificaciones cuantas veces deseemos.**

```
PROGRAMA modifica
ENTORNO:
res <- "S"
ALGORITMO:
MIENTRAS res = "S"HACER
ABRIR "DATOS.DAT"
sw <- 0
num <- 0
nom <- Espacios( 15 )
ape <- Espacios( 30 )
dir <- Espacios( 20 )
pro <- Espacios( 20 )
```

ISBN: 978-631-6960-03-0

```
Borrar_pantalla( )
EN 5,10 ESCRIBIR "D.N.I.: "
EN 5,18 LEER num
MIENTRAS NO Eof( ) Y sw = 0 HACER
SI dni = num ENTONCES
HACER Imprimir
HACER Cambios
sw <- 1
SINO
Siguiente_registro( )
FINSI
FINMIENTRAS
SI sw = 0 ENTONCES
HACER Detener
FINSI
CERRAR "DATOS.DAT"
res <- Espacios( 1 )
HACER Mas
FINMIENTRAS
FINPROGRAMA
------------------------------
SUBPROGRAMA Mas
MIENTRAS res <> "S"Y res <> "N"HACER
ESCRIBIR "Desea m s cambios (S/N): "
LEER res
res <- Convertir_mayusculas( res )
FINMIENTRAS
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Imprimir
EN 7,5 ESCRIBIR "Nombre: "
EN 9,5 ESCRIBIR .Apellidos: "
EN 11,5 ESCRIBIR "Direcci&oacute;n: "
```

ISBN: 978-631-6960-03-0

```
EN 13,5 ESCRIBIR "Provincia: "
EN 7,16 LEER nombre
EN 9,16 LEER apellidos
EN 11,16 LEER direccion
EN 13,16 LEER provincia
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Cambios
nom <- nombre
ape <- apellidos
dir <- direccion
pro <- provincia
EN 7,16 LEER nom
EN 9,16 LEER ape
EN 11,16 LEER dir
EN 13,16 LEER pro
nombre <- nom
apellidos <- ape
direccion <- dir
provincia <- pro
Escribir_registro( )
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Detener
EN 10,20 ESCRIBIR Registro inexistente"
EN 20,18 ESCRIBIR "Pulse INTRO para continuar"
Pausa( )
FINSUBPROGRAMA
```

46. **Tenemos el fichero secuencial DATOS.DAT cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA. Listar por impresora todos los registros cuya provincia sea una determinada que introduciremos por teclado.**

```
PROGRAMA provincia
```

ISBN: 978-631-6960-03-0

```
ENTORNO:
fi <- 55
c <- 0
pag <- 1
pro <- Espacios( 15 )
ALGORITMO:
Borrar_pantalla( )
EN 10,20 ESCRIBIR "Provincia: "
EN 10,32 LEER pro
ABRIR "DATOS.DAT"
Activar_impresora( )
MIENTRAS NO Eof( ) HACER
SI provincia = pro ENTONCES
SI fi = 55 ENTONCES
HACER Cabecera
FINSI
EN fi,5 ESCRIBIR dni
EN fi,15 ESCRIBIR nombre
EN fi,35 ESCRIBIR apellidos
EN fi,65 ESCRIBIR direccion
fi <- fi + 1
c <- c + 1
FINSI
Siguiente_registro( )
FINMIENTRAS
SI pag <> 1 ENTONCES
EN fi + 2,20 ESCRIBIR "Total de personas: "
EN fi + 2,39 ESCRIBIR c
FINSI
Activar_pantalla( )
CERRAR "DATOS.DAT"
FINPROGRAMA
------------------------------
```

ISBN: 978-631-6960-03-0

```
SUBPROGRAMA Cabecera
Salto_pagina( )
EN 2,65 ESCRIBIR "P g.: "
EN 2,71 ESCRIBIR pag
EN 4,10 ESCRIBIR Relaci¢n de las personas que viven en la provincia: "
EN 4,62 ESCRIBIR pro
EN 6,7 ESCRIBIR "D.N.I."
EN 6,18 ESCRIBIR "Nombre"
EN 6,40 ESCRIBIR .ªpellidos"
EN 6,68 ESCRIBIR "Direcci¢n"
EN 7,4 ESCRIBIR ----------------------------------------------------------------------"
fi <- 9
pag <- pag + 1
FINSUBPROGRAMA
```

47. **En el fichero secuencial VENTAS.DAT, están almacenadas las ventas de los productos durante el día, cuyos campos son: NART y VENTAS. Se desea hacer un programa que liste por impresora todas las ventas realizadas durante el día.**

```
PROGRAMA ventas
ENTORNO:
total <- 0
uno <- 0
fi <- 55
sw <- 0
aux <- 0
pag <- 1
ALGORITMO:
Borrar_pantalla( )
Activar_impresora( )
ABRIR "VENTAS.DAT"
MIENTRAS NO Eof( ) HACER
SI fi = 55 ENTONCES
```

ISBN: 978-631-6960-03-0

```
HACER Cabecera
FINSI
SI sw = 0 ENTONCES
aux <- nart
sw <- 1
FINSI
SI nart = aux ENTONCES
uno <- uno + ventas
SINO
HACER Imprimir
uno <- 0
aux <- nart
uno <- ventas
FINSI
Siguiente_registro( )
FINMIENTRAS
HACER Imprimir
EN fi + 2,20 ESCRIBIR Ünidades vendidas: "
EN fi + 2,39 ESCRIBIR total
Activar_pantalla( )
CERRAR "VENTAS.DAT"
FINPROGRAMA
------------------------------
SUBPROGRAMA Imprimir
EN fi,32 ESCRIBIR aux
EN fi,42 ESCRIBIR total
fi <- fi + 1
total <- total + uno
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Cabecera
Salto_pagina( )
EN 2,65 ESCRIBIR "P g.: "
```

ISBN: 978-631-6960-03-0

```
EN 2,71 ESCRIBIR pag
EN 4,20 ESCRIBIR "LISTADO DE LAS VENTAS DE LOS PRODUCTOS AL DIA:
"
EN 4,68 ESCRIBIR Fecha_sistema( )
EN 6,30 ESCRIBIR "N£mero"
EN 6,40 ESCRIBIR Çantidad"
EN 7,18 ESCRIBIR ------------------------------------------------"
fi <- 9
pag <- pag + 1
FINSUBPROGRAMA
```

48. **Dado el fichero secuencial ARTICULOS.DAT, cuyos campos son: NART, ARTÍCULO, PVP, STOCK y MÍNIMO. En otro fichero VENTAS.DAT, están almacenadas las modificaciones de los productos durante el día, cuyos campos son: NART2, VENTAS y TIPO. El campo TIPO puede tomar los valores 0 (venta) y 1 (compra). Se desea hacer un programa que realice una actualización del fichero de ARTICULOS y un listado por impresora de las entradas y salidas de los artículos.**

```
PROGRAMA modifica
ENTORNO:
entra <- 0
sale <- 0
total <- 0
fi <- 55
sw <- 0
aux <- 0
pag <- 1
ALGORITMO:
Borrar_pantalla( )
Activar_impresora( )
ABRIR .ARTICULOS.DAT"
Primer_registro( )
ABRIR "SALIDAS.DAT"
```

ISBN: 978-631-6960-03-0

```
Primer_registro( )
ABRIR "VENTAS.DAT"
Primer_registro( )
SELECCIONAR "VENTAS.DAT"
MIENTRAS NO Eof( ) HACER
SI fi = 55 ENTONCES
HACER Cabecompras
FINSI
SI sw = 0 ENTONCES
aux <- nart2
HACER Buscar
sw <- 1
FINSI
SI nart2 = aux ENTONCES
HACER Calculos
SINO
HACER Grabar
HACER Compra
entra <- 0
sale <- 0
aux <- nart2
HACER Buscar
HACER Calculos
FINSI
Siguiente_registro( )
FINMIENTRAS
HACER Grabar
HACER Compra
SELECCIONAR "SALIDAS.DAT"
fi <- 55
MIENTRAS NO Eof( ) HACER
SI fi = 55 ENTONCES
HACER Cabesal
```

ISBN: 978-631-6960-03-0

```
FINSI
MIENTRAS nart3 <> nart HACER
SELECCIONAR .ARTICULOS.DAT"
Siguiente_registro( )
FINMIENTRAS
aux <- nart3
HACER Buscar
HACER Sale
SELECCIONAR "SALIDAS.DAT"
Siguiente_registro( )
FINMIENTRAS
EN fi + 4,55 ESCRIBIR "Total: "
EN fi + 4,62 ESCRIBIR total
Activar_pantalla( )
Cerrar_ficheros( )
BORRAR "SALIDAS.DAT"
FINPROGRAMA
------------------------------
SUBPROGRAMA Calculos
SI tipo = 0 ENTONCES
entra <- entra + ventas
SINO
sale <- sale + ventas
FINSI
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Grabar
stock <- stock + entra - sale
SELECCIONAR .ARTICULOS.DAT"
Escribir_registro( )
nart3 <- aux
ventas3 <- sale
SELECCIONAR "SALIDAS.DAT"
```

ISBN: 978-631-6960-03-0

```
Final_fichero( )
Escribir_registro( )
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Cabecompras
Salto_pagina( )
EN 2,65 ESCRIBIR "P g.: "
EN 2,71 ESCRIBIR pag
EN 4,20 ESCRIBIR "LISTADO DE LAS ENTRADAS DE ARTICULOS AL DIA: "
EN 4,66 ESCRIBIR Fecha_sistema( )
EN 6,18 ESCRIBIR "N£mero"
EN 6,35 ESCRIBIR .ªrt¡culo"
EN 6,65 ESCRIBIR Çantidad"
EN 7,15 ESCRIBIR ----------------------------------------------------------------"
fi <- 9
pag <- pag + 1
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Compra
EN fi,16 ESCRIBIR aux
EN fi,30 ESCRIBIR articulo
EN fi,67 ESCRIBIR entra
fi <- fi + 1
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Cabesal
Salto_pagina( )
EN 2,65 ESCRIBIR "P g.: "
EN 2,71 ESCRIBIR pag
EN 4,20 ESCRIBIR "LISTADO DE SALIDAS DE ARTICULOS AL DIA: "
EN 4,60 ESCRIBIR Fecha_sistema( )
EN 6,8 ESCRIBIR "N£mero"
EN 6,16 ESCRIBIR .ªrt¡culo"
```

ISBN: 978-631-6960-03-0

```
        EN 6,40 ESCRIBIR Çantidad"
        EN 6,54 ESCRIBIR "PVP"
        EN 6,64 ESCRIBIR Ïmporte"
        EN 7,6 ESCRIBIR ----------------------------------------------------------------------------
-"
        fi <- 9
        pag <- pag + 1
        FINSUBPROGRAMA
        ------------------------------
        SUBPROGRAMA Sale
        Salto_pagina( )
        SI stock < minimo ENTONCES
        EN fi,4 ESCRIBIR "*"
        FINSI
        EN fi,6 ESCRIBIR nart3
        EN fi,14 ESCRIBIR articulo
        EN fi,40 ESCRIBIR ventas
        EN fi,54 ESCRIBIR pvp
        EN fi,65 ESCRIBIR ventas * pvp
        total <- total + ventas * pvp
        fi <- fi + 1
        FINSUBPROGRAMA
        ------------------------------
        SUBPROGRAMA Buscar
        MIENTRAS nart <> aux HACER
        SELECCIONAR .ARTICULOS.DAT"
        Siguiente_registro( )
        FINMIENTRAS
        FINSUBPROGRAMA
```

ISBN: 978-631-6960-03-0

12.9 TEMA 9: Organización aleatoria y secuencial

49. **Hacer un pseudocódigo que nos permita dar altas en el fichero DATOS.DAT de organización directa, controlando las altas duplicadas. Los campos son: DNI, NOMBRE, APELLIDOS Y PUNTERO para ambos archivos.**

Algoritmo(dn) =

Blanco: grabamos el registro en esa posici¢n y ponemos el puntero a cero**.**

Cero: comprobamos cu l es el valor del campo puntero. Si es cero, grabamos el registro en esa posici¢n (no hay sinonimos) y si es distinto de cero, comparamos el valor con el campo DNI, si son iguales, alta duplicada y dejamos de leer, y si no son iguales, introducimos el resto de la informacion.

Distinto de cero: hay un registro grabado en esa posicion. Si es igual al dato introducido, alta duplicada, y si no son iguales, comprobamos el valor del puntero, si es cero grabamos el registro, y si no es cero, si es igual al campo DNI, alta duplicada y sino se graba la informacion enel archivo SINONIMOS.DAT.

PROGRAMA altas
ENTORNO:
res <- "S"
ALGORITMO:
ABRIR "DATOS.DAT"
ABRIR "SINONIMOS.DAT"
MIENTRAS res = "S"HACER
dn <- 0
nom <- Espacios(15)
ape <- Espacios(30)
dir <- Espacios(35)
aux <- Espacios(2)
hueco <- Espacios(2)
swh <- 0
sw <- 0
num <- 0
donde <- 0
i <- 0

ISBN: 978-631-6960-03-0

```
Borrar_pantalla( )
EN 10,20 ESCRIBIR "D.N.I.: "
EN 10,29 LEER dn
num <- Algoritmo( dn )
SELECCIONAR "DATOS.DAT"
LEER num
SI dni = Str( dn ) ENTONCES
HACER Alta_duplicada
SINO
SI Val( dni ) = 0 O dni = Espacios( ) ENTONCES
swh <- 1
FINSI
SI Val( puntero ) = 0 O puntero = Espacios( ) ENTONCES
HACER Introducir
puntero <- Str( 0 )
GRABAR num
SINO
HACER Buscar
SI sw = 0 ENTONCES
HACER Introducir
SI swh = 1 ENTONCES
GRABAR num
SINO
HACER Encontrar_sitio
SELECCIONAR "SINONIMOS.DAT"
GRABAR 1
puntero <- Str( donde )
SI i = 0 ENTONCES
SELECCIONAR "DATOS.DAT"
GRABAR num
SINO
SELECCIONAR "SINONIMOS.DAT"
GRABAR Val( aux )
```

ISBN: 978-631-6960-03-0

```
FINSI
puntero <- Str( 0 )
SELECCIONAR "SINONIMOS.DAT"
GRABAR donde
FINSI
SINO
HACER Alta_duplicada( )
FINSI
FINSI
FINSI
HACER Mas
FINMIENTRAS
Cerrar_ficheros( )
FINPROGRAMA
------------------------------
SUBPROGRAMA Introducir
EN 12,20 ESCRIBIR "Nombre: "
EN 14,20 ESCRIBIR .ªpellidos: "
EN 16,20 ESCRIBIR "Direcci¢n: "
EN 12,29 LEER nom
EN 14,32 LEER ape
EN 16,32 LEER dir
FINSUBPROGRAMA
------------------------------
SUBPROGRAMA Encontrar_sitio
SELECCIONAR "SINONIMOS.DAT"
LEER 1
SI Val( nombre ) <> 0 Y nombre <> Espacios( ) ENTONCES
donde <- Val( nombre )
LEER donde
hueco <- Val( nombre )
nombre <- Str( hueco )
SINO
```

ISBN: 978-631-6960-03-0

```
donde <- Val( dni ) + 1
dni <- Str( donde )
FINSI
FINSUBPROGRAMA
-----------------------------
SUBPROGRAMA Buscar
aux <- puntero
i <- 0
MIENTRAS Val( puntero ) <> 0 Y sw = 0 HACER
SELECCIONAR "SINONIMOS.DAT"
LEER Val( puntero )
SI dni = Str( dn ) ENTONCES
EN 20,10 ESCRIBIR ."Alta duplicada"
Pausa( )
sw <- 1
SINO
SI Val( puntero ) <> 0 ENTONCES
i <- i + 1
aux <- puntero
FINSI
FINSI
FINMIENTRAS
FINSUBPROGRAMA
-----------------------------
SUBPROGRAMA Alta_duplicada
EN 20,10 ESCRIBIR ."Alta duplicada"
Pausa( )
FINSUBPROGRAMA
```

50. **Tenemos el fichero DATOS.DAT, que está indexado por el campo APELLIDOS, cuyos campos son: DNI, NOMBRE, APELLIDOS, DIRECCIÓN y PROVINCIA. Hacer un programa que nos permita listar por pantalla todos los registros del fichero, controlando el salto de página cuando llegue a la línea veinte.**

ISBN: 978-631-6960-03-0

```
PROGRAMA listar
ENTORNO:
fi <- 22
ALGORITMO:
ABRIR "DATOS.DATÏNDICE .ªPELLIDO"
MIENTRAS NO Eof( ) HACER
SI fi = 22 ENTONCES
HACER Cabecera
FINSI
EN fi,2 ESCRIBIR dni
EN fi,12 ESCRIBIR nombre
EN fi,28 ESCRIBIR apellidos
EN fi,55 ESCRIBIR direccion
EN fi,69 ESCRIBIR provincia
fi <- fi + 1
SI fi = 20 ENTONCES
EN 22,20 ESCRIBIR "Pulse INTRO para continuar"
Pausa( )
fi <- 22
FINSI
Siguiente_registro( )
FINMIENTRAS
CERRAR "DATOS.DAT"
Cerrar_indices( )
FINPROGRAMA
------------------------------
SUBPROGRAMA Cabecera
Borrar_pantalla( )
EN 3,4 ESCRIBIR "D.N.I."
EN 3,20 ESCRIBIR "NOMBRE"
EN 3,35 ESCRIBIR .ªPELLIDOS"
EN 3,60 ESCRIBIR "DIRECCION"
EN 3,70 ESCRIBIR "PROVINCIA"
```

ISBN: 978-631-6960-03-0

```
fi <- 5
FINSUBPROGRAMA
```

51. **Tenemos el fichero DATOS.DAT con la misma estructura anterior, que está indexado por el campo DNI. Crear un programa que nos permita consultar un registro siempre que queramos.**

```
PROGRAMA consulta
ENTORNO:
res <- "S"
ALGORITMO:
ABRIR "DATOS.DATÏNDICE "DNI"
MIENTRAS res = "S"HACER
num <- 0
Borrar_pantalla( )
EN 8,20 ESCRIBIR "D.N.I. a buscar: "
EN 8,38 LEER num
BUSCAR num
SI Encontrado( ) ENTONCES
EN 10,12 ESCRIBIR "Nombre: ", nombre
EN 12,28 ESCRIBIR .ªpellidos: ", apellidos
EN 14,55 ESCRIBIR "Direcci¢n: ", direccion
EN 16,69 ESCRIBIR "Provincia: ", provincia
SINO
EN 12,20 ESCRIBIR "No est "
EN 16,20 ESCRIBIR "Pulse INTRO para continuar"
Pausa( )
FINSI
res <- Espacios( 1 )
HACER Mas
FINMIENTRAS
CERRAR "DATOS.DAT"
Cerrar_indices( )
FINPROGRAMA
```

ISBN: 978-631-6960-03-0

REFERENCIAS BIBLIOGRÁFICAS

Advance, por R. C. (2025). *GUI Parte 1/ Diseñar Formulario Moderno + Sub Menú con C# y Windows Form – Versión Básica (Beta) – RJ Code Advance*. https://rjcodeadvance.com/disenar-interfaz-grafico-de-usuario-moderno-con-c-y-windows-form/

Aguilar, L. J., Azuela, M. F., & Baena, L. R. (1996). *Fundamentos de programaci¢n: Libro de problemas*. McGraw Hill.

Aguilar, L. J., López, A. H., & Martínez, I. Z. (1994). *Pascal y Turbo Pascal: Un Enfoque Práctico*. McGraw-Hill/Interamericana de Espana, S.A.

ANDREWS, M. (1997). *APRENDA VISUAL C++ YA-ANDREWS* (M. D. B. RODRIGUEZ, Trad.).

Beúnes Cañete, J. E., Vargas Ricardo, A., Beúnes Cañete, J. E., & Vargas Ricardo, A. (2019). La introducción de la herramienta didáctica PSeInt en el proceso de enseñanza aprendizaje: Una propuesta para Álgebra Lineal. *Transformación*, *15*(1), 147-157.

Bores Rangel, M. del R., & Rosales Becerril, R. (1993). *Computación, metodología, lógica computacional y programación*. McGraw-Hill.

Cairó Battistutti, O. (2005). *Metodología de la programación: Algoritmos, diagramas de flujo y programas* (3. ed). Alfaomega.

Chapra, S. C. (with Canale, R. P.). (1989). *Introducción a la computación para ingenieros*. McGraw-Hill.

Copi, I. M. (1998). *Lógica simbólica*. Compañía Editorial Continental. https://books.google.com.ec/books?id=zZ--AAAACAAJ

Deitel, H. M., & Deitel, P. J. (2003). *Cómo programar en C++* (Cuarta edición). Pearson Educación.

Duglas, O. (2013, marzo 28). Breve reseña histórica del lenguaje de programación en C. *La web de la programación*. https://duglasm.wordpress.com/tutoriales-de-progrmacion/tutorial-del-lenguaje-de-programacion-c/breve-resena-historica-del-lenguaje-de-programacion-en-c/

Gayan Segarra, J. D. (1988). *Curso General de Informática*. Gustavo Gili S.A.

Gottfried, B. S. (1997). *Programación en C*. McGraw-Hill/Interamericana de Espana, S.A.

Gutiérrez, R. G. (2011, noviembre 17). Consultas en SQL II. *Sistemasumma.com*. https://sistemasumma.com/2011/11/17/consultas-en-sql-ii/

INTEF. (2025). PSeInt: Programando en pseudocódigo. *INTEF*. https://intef.es/observatorio_tecno/pseint-programando-en-pseudocodigo/

Irving M. (1979). *Lógica Simbólica*. C.E.C.S.A.

Joyanes Aguilar, L. (1996). *Fundamentos de Programación. Algoritmos y estructuras de datos* (2.ª ed.).

ISBN: 978-631-6960-03-0

McGraw-Hill.

Leestma, S. (1999). *Programación en pascal iv edición* (4a.ed.). Prentice Hall.

Marketing. (2025, febrero 7). Lenguajes de programación | MSMK University. *MSMK*. https://msmk.university/las-generaciones-de-lenguajes-de-programacion-1gl-2gl-3gl-4gl-y-5gl/

OverIQ. (2020, junio 27). *Character Array and Character Pointer in C*. OverIQ.Com. https://overiq.com/c-programming-101/character-array-and-character-pointer-in-c/

Pappas, C. H., Murray, W. H., Papas, C. H., & Murray, W. (s. f.). *Microsoft Visual C ++ 6.0 Manual De Referencia*.

Pineda, M., Bonilla, J., & Poveda, D. (2025). *Prolog by mppinedav*. https://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/index.html

pseint. (2025). *PSeInt—Diagrama N-S (Nassi-Schneiderman)*. https://pseint.sourceforge.net/index.php?page=ejemplos.php&cual=Potencia&mode=ns

Pseintlab. (2024, noviembre 9). *Ejercicios con estructura repetitiva Para/Desde en Pseint | PSeIntLab*. https://pseintlab.com/ejercicios-estructura-repetitiva-para-pseint/

Rahman, M. M., Watanobe, Y., Matsumoto, T., Kiran, R. U., & Nakamura, K. (2022). Educational Data Mining to Support Programming Learning Using Problem-Solving Data. *Ieee Access*, *10*, 26186-26202. https://doi.org/10.1109/ACCESS.2022.3157288

Sebesta, R. W. (2012). *Concepts of programming languages* (10. ed). Pearson.

Sierra, F. J. C. (1997). *Enciclopedia del Lenguaje C*.

Tremblay, J.-P. (with Bunt, R. B.). (1982). *Introducción a la ciencia de las computadoras: Enfoque algorítmico*. McGraw-Hill de México.

tutorialhorizon. (2026). *Linear Search vs Binary Search*. https://www.tutorialhorizon.com/algorithms/linear-search-vs-binary-search

Urbina, C. (2012, febrero 19). Cecilia Urbina: Aplicaciones del lenguaje ensamblador. *Cecilia Urbina*. https://cecilia-urbina.blogspot.com/2012/02/aplicaciones-del-lenguaje-ensamblador.html

Ureña López, L. A. (1997). Fundamentos de informática / L.Alfonso Ureña López... [Et al.]. En *Fundamentos de informática*. RAMA.

Ureña López, L. A. (1999). *Fundamentos de informatica*. Alfaomega.

ISBN: 978-631-6960-03-0

DE LOS AUTORES

Teresita Lourdes Argüello Estrella

Magisterio Nacional Ecuatoriano. 020101, Guaranda, Bolívar, Ecuador.

teresita.arguello@docentes.educacion.edu.ec

https://orcid.org/0009-0002-1428-6719

Fabian Eduardo Fierro Saltos

Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.

fabian.fierro@ueb.edu.ec

https://orcid.org/0009-0002-8698-5930

Enrique Marcelo Baño León

Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.

enrique.bano@ueb.edu.ec

https://orcid.org/0000-0002-5550-0649

Galuth Irene García Camacho

Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.

ggarcia@ueb.edu.ec

https://orcid.org/0000-0001-8692-4017

Patricia Isabel Albán Yanez

Universidad Estatal de Bolívar. 020101, Guaranda, Bolívar, Ecuador.

paalban@ueb.edu.ec

https://orcid.org/0000-0003-3799-4195

Edwin Patricio Avilés Meza

Universidad Estatal de Milagro. 091050, Milagro, Manabí, Ecuador.

eavilesm6@unemi.edu.ec

https://orcid.org/0009-0002-4767-3424

PUERTO MADERO
EDITORIAL

ISBN 978-631-6960-03-0