# I. Introducción a la Computación

## 1. Elementos de un sistema de cómputo

Un sistema de cómputo está compuesto por varios elementos fundamentales que interactúan entre sí para realizar tareas específicas.

### Hardware:

Definición: Los componentes físicos del sistema de cómputo.

Ejemplos: CPU, memoria RAM, discos duros, periféricos de entrada (teclado, ratón) y salida (monitor, impresora).

### Software:

Definición: Programas y sistemas operativos que controlan el hardware y realizan tareas específicas.

Ejemplos: Microsoft Windows, Linux, aplicaciones como Word o Excel.

### Usuarios:

Definición: Personas que interactúan con el sistema para realizar tareas.

Ejemplos: Programadores, diseñadores gráficos, estudiantes.

### Datos:

Definición: Información que se procesa y almacena.

Ejemplos: Documentos, imágenes, bases de datos.

### Redes:

Definición: Infraestructura que conecta múltiples sistemas de cómputo.

Ejemplos: Wi-Fi, Ethernet, Internet.

### Infraestructura energética:

Definición: Elementos que garantizan el suministro de energía al sistema.

Ejemplos: Fuentes de poder, UPS, generadores eléctricos.

## 2. Lenguajes de programación: Definición y clasificación

Los lenguajes de programación son herramientas esenciales para desarrollar software.

Definición: Conjunto de instrucciones que los programadores utilizan para comunicarse con las computadoras.

### Clasificación:

Lenguajes de bajo nivel:

- Operan cerca del hardware y son específicos de la arquitectura de la máquina.

- Ejemplos: Ensamblador, lenguaje máquina.

Lenguajes de alto nivel:

- Más abstractos, fáciles de leer y escribir.

- Ejemplos: Python, Java, C++.

Paradigmas de programación:

Imperativo: Secuencia de instrucciones (C, Python).

Orientado a objetos: Basado en clases y objetos (Java, C).

Funcional: Enfocado en funciones matemáticas (Haskell, Lisp).

Lógico: Basado en reglas y lógica formal (Prolog).

## 3. Herramientas de programación

Herramientas que facilitan el desarrollo, mantenimiento y ejecución de software.

### Editores

Definición: Aplicaciones para escribir y editar código fuente.

Ejemplos:

- Visual Studio Code: Soporte para múltiples lenguajes y extensiones.

- Sublime Text: Ligero y rápido.

- Notepad++: Ideal para tareas básicas.

### Compiladores

Definición: Transforman el código fuente en un formato que la computadora pueda ejecutar directamente.

Ejemplos:

- GCC: Compilador para C y C++.

- Clang: Alternativa moderna a GCC.

### Encadenadores

Los encadenadores (o linkers, en inglés) son herramientas de software que desempeñan un papel crucial en el proceso de compilación de programas. Su función principal es enlazar o combinar múltiples archivos de código objeto (archivos .obj o .o) en un solo archivo ejecutable o biblioteca compartida. Este proceso permite que los programas escritos en lenguajes de alto nivel puedan ejecutarse en un sistema operativo, ya que transforma el código objeto en una forma que puede ejecutarse directamente.

Definición

Los encadenadores son herramientas de desarrollo que toman como entrada archivos de código objeto (los cuales son productos de la compilación de archivos fuente) y los combinan para generar un archivo ejecutable o una biblioteca. Los encadenadores también se encargan de resolver las dependencias entre los diferentes módulos del programa, asegurando que todas las referencias a funciones y variables sean correctamente resueltas y apuntadas a las ubicaciones adecuadas.

Funciones principales de los encadenadores:

* Resolución de símbolos: Verifica y enlaza las referencias a funciones o variables externas en el código objeto. Si una función está definida en un archivo objeto y llamada desde otro, el enlazador asegura que la llamada se redirija a la ubicación correcta en el ejecutable.
* Combinación de archivos objeto: Junta varios archivos de código objeto en un solo archivo ejecutable.
* Asignación de direcciones de memoria: Asigna direcciones de memoria a las variables y funciones en el ejecutable.
* Enlace con bibliotecas externas: Incorpora bibliotecas preexistentes (estáticas o compartidas) en el ejecutable para proporcionar funcionalidad adicional (como funciones de manejo de archivos o manipulación de cadenas).

Ejemplos de encadenadores:

* GCC (GNU Compiler Collection): GCC incluye un enlazador llamado ld (el GNU Linker). Este enlazador es una herramienta poderosa y flexible que se usa comúnmente en sistemas basados en Unix/Linux para enlazar archivos de código objeto y crear ejecutables.
* Visual Studio (Microsoft): Visual Studio utiliza el Linker de Microsoft (por ejemplo, link.exe) para realizar el proceso de enlace de archivos objeto y bibliotecas en un ejecutable o DLL. Este enlazador también incluye características avanzadas para la gestión de dependencias y optimización de código.

Ejemplo de uso de un enlazador:

Supongamos que tienes dos archivos fuente en C, main.c y utils.c, y los compilas para obtener archivos de código objeto main.o y utils.o. El enlazador los combinará para formar un ejecutable llamado programa.exe que puedes ejecutar.

Comando de compilación y enlace usando GCC:

bash

Copiar código

gcc main.c utils.c -o programa

Proceso:

1. gcc compila main.c y utils.c en archivos de código objeto (main.o y utils.o).
2. El enlazador se ejecuta automáticamente y enlaza main.o y utils.o en un ejecutable llamado programa.

Los encadenadores también pueden manejar archivos de bibliotecas, por ejemplo, bibliotecas estáticas (.a) o bibliotecas compartidas (.so o .dll), que contienen funciones que el programa puede usar. Esto permite al programa reutilizar código que no necesita estar incluido directamente en el archivo ejecutable.

En resumen, los encadenadores son herramientas esenciales en el desarrollo de software que permiten la creación de ejecutables funcionales al combinar y resolver todas las referencias entre archivos de código objeto y bibliotecas.

### Depuradores

Depurador: Es un tipo de software que permite a los programadores monitorear la ejecución de un programa paso a paso, examinar el contenido de las variables, evaluar expresiones y controlar el flujo de ejecución. El uso de un depurador facilita la localización y corrección de errores (bugs) en el código, haciendo el proceso de desarrollo más eficiente y efectivo.

Ejemplos de Depuradores

1. GDB (GNU Debugger):
   * Definición: GDB es un depurador de código abierto para programas escritos en C, C++, y otros lenguajes de programación. Es ampliamente utilizado en el desarrollo de software de sistemas y aplicaciones.
   * Características:
     + Permite detener la ejecución de un programa en puntos específicos (puntos de interrupción).
     + Permite ejecutar el programa paso a paso.
     + Ofrece la capacidad de inspeccionar y modificar el valor de las variables en tiempo de ejecución.
     + Es compatible con la depuración de programas en ejecución y la depuración remota.
   * Ejemplo de uso:

bash

Copiar código

gdb programa

Luego, se puede usar comandos como run, break, next, print, etc., para controlar y examinar la ejecución.

1. Visual Studio Debugger:
   * Definición: Es el depurador integrado en el entorno de desarrollo integrado (IDE) de Visual Studio, utilizado para depurar aplicaciones desarrolladas en C#, C++, Visual Basic .NET, y otros lenguajes de programación compatibles.
   * Características:
     + Ofrece una interfaz gráfica para establecer puntos de interrupción y controlar la ejecución del programa.
     + Permite inspeccionar el estado de las variables, pilas de llamadas y objetos.
     + Ofrece opciones avanzadas como la evaluación de expresiones y la ejecución condicional.
     + Se integra con otras herramientas de Visual Studio para un análisis más completo.
   * Ejemplo de uso:
     + Para usar el depurador en Visual Studio, basta con presionar F5 (iniciar depuración) y F9 para establecer puntos de interrupción en el código.

Ejemplo de Depuración en Java (con un Depurador de IDE)

Si estás programando en Java, puedes usar depuradores integrados en IDEs como IntelliJ IDEA, Eclipse, o NetBeans. Aquí hay un ejemplo de cómo usar el depurador en IntelliJ IDEA:

1. Establece un punto de interrupción: Haz clic en el margen izquierdo del editor de código en la línea donde quieres que la ejecución se detenga.
2. Inicia la depuración: Haz clic en el botón de depuración (ícono de un insecto) o presiona Shift + F9.
3. Inspecciona variables y controla la ejecución: Usa las herramientas del depurador para ver los valores de las variables y avanzar paso a paso a través del código.

Importancia de los Depuradores

* Detección de errores: Permiten encontrar errores lógicos o de ejecución en el programa.
* Optimización del código: Ayudan a comprender cómo se ejecuta el código y cómo se utilizan los recursos, permitiendo realizar mejoras.
* Pruebas más eficientes: Facilitan la realización de pruebas más detalladas y específicas.

Conclusión

Los depuradores son herramientas esenciales para el desarrollo de software, ya que ayudan a detectar y corregir errores de manera más rápida y eficiente. GDB y el depurador de Visual Studio son ejemplos de depuradores ampliamente utilizados en sus respectivos ámbitos de desarrollo. Utilizar un depurador adecuadamente puede ahorrar tiempo y mejorar la calidad del software.

### Administradores de versiones

Los **administradores de versiones** son herramientas esenciales en el desarrollo de software moderno, ya que permiten llevar un registro de los cambios realizados en el código fuente de un proyecto a lo largo del tiempo. Estos sistemas no solo ayudan a mantener un historial de cambios, sino que también facilitan la colaboración entre desarrolladores y la gestión de distintas versiones de un proyecto.

**Definición de Administradores de Versiones**

**Administradores de versiones**: Son herramientas que permiten a los desarrolladores registrar, controlar y gestionar los cambios realizados en el código fuente de un proyecto. Estos sistemas ayudan a mantener un historial detallado de cada modificación, permitiendo volver a versiones anteriores, comparar versiones y colaborar de manera eficiente con otros miembros del equipo.

**Ejemplos de Administradores de Versiones**

1. **Git**:
   * **Definición**: Git es un sistema de control de versiones distribuido, lo que significa que cada desarrollador tiene una copia completa del historial del proyecto en su máquina local. Es ampliamente utilizado en proyectos de código abierto y se ha convertido en el estándar de facto para el control de versiones.
   * **Características**:
     + **Descentralizado**: Cada desarrollador tiene su propio repositorio local, lo que permite trabajar de manera independiente y sincronizar los cambios posteriormente.
     + **Rápido y eficiente**: Git está diseñado para manejar grandes proyectos con rapidez y eficiencia.
     + **Soporte para ramas**: Facilita la creación y gestión de ramas para el desarrollo paralelo de nuevas características y la corrección de errores.
     + **Integración con plataformas de colaboración**: Git es compatible con plataformas como GitHub, GitLab y Bitbucket, lo que facilita la colaboración entre equipos.
   * **Ejemplo de uso básico**:

bash

Copiar código

git init # Inicializa un nuevo repositorio Git

git add archivo.txt # Agrega un archivo al área de preparación

git commit -m "Mensaje de commit" # Realiza un commit con un mensaje

git status # Muestra el estado de los archivos

git log # Muestra el historial de commits

git branch # Lista las ramas en el repositorio

git checkout rama # Cambia de rama

1. **SVN (Subversion)**:
   * **Definición**: SVN es un sistema de control de versiones centralizado que permite a los desarrolladores mantener un repositorio centralizado en un servidor. Aunque ha sido superado por sistemas distribuidos como Git, sigue siendo utilizado en ciertos proyectos y organizaciones.
   * **Características**:
     + **Centralizado**: Todos los desarrolladores trabajan con un único repositorio central, lo que requiere conexión al servidor para realizar operaciones.
     + **Control de versiones lineal**: Los cambios se registran de manera secuencial, lo que facilita la gestión de versiones en algunos casos.
     + **Facilidad de uso**: Aunque es más simple que Git, SVN es adecuado para proyectos más pequeños o para equipos que prefieren un enfoque centralizado.
   * **Ejemplo de uso básico**:

svn checkout https://example.com/repositorio # Clona el repositorio

svn add archivo.txt # Agrega un archivo al repositorio

svn commit -m "Mensaje de commit" # Realiza un commit con un mensaje

svn update # Actualiza tu copia de trabajo con los últimos cambios

svn status # Muestra el estado de los archivos

**Beneficios de Usar Administradores de Versiones**

* **Historial de cambios**: Permiten mantener un registro de cada cambio realizado, facilitando la revisión de la evolución del proyecto.
* **Colaboración**: Los equipos de desarrollo pueden trabajar en paralelo, realizar revisiones y fusionar cambios de manera eficiente.
* **Desarrollo paralelo**: La creación de ramas permite desarrollar nuevas funcionalidades o corregir errores sin afectar la versión principal del proyecto.
* **Recuperación de versiones anteriores**: Es posible revertir cambios o recuperar versiones anteriores del proyecto en caso de errores o pérdidas de datos.
* **Resolución de conflictos**: Los administradores de versiones modernos tienen herramientas para resolver conflictos que pueden surgir cuando varios desarrolladores modifican el mismo archivo.

**Conclusión**

Los **administradores de versiones** son herramientas clave para el desarrollo de software moderno, ayudando a organizar y gestionar cambios en el código de manera eficiente. Git y SVN son ejemplos populares de estos sistemas, cada uno con sus características y casos de uso particulares. Git, por su naturaleza distribuida y su amplia adopción, es la opción más popular en la actualidad, mientras que SVN todavía se usa en proyectos donde la centralización es preferida. La elección entre Git y SVN dependerá de las necesidades específicas de tu proyecto y equipo.

### Generadores de código

Definición: Herramientas que automatizan la creación de código repetitivo.

Ejemplos:

- Yeoman: Generador de proyectos front-end.

- JHipster: Generador para aplicaciones web.

Ambientes operativos

Definición: Entornos donde se desarrollan y ejecutan programas.

Ejemplos:

- Docker: Contenedores para aplicaciones.

- Entornos virtuales en Python.

# II. Introducción a Algoritmos

## 1. Concepto de algoritmo, características y componentes

### Algoritmo:

Definición: Secuencia de pasos organizados y finitos para resolver un problema.

Ejemplo: Algoritmo para calcular el promedio de tres números.

### Características:

- Finito: Debe terminar en algún punto.

- Preciso: Cada paso debe estar definido claramente.

- Definido: Producirá el mismo resultado bajo las mismas condiciones.

### Componentes:

- Entrada: Datos iniciales.

- Proceso: Transformación de datos.

- Salida: Resultado final.

## 2. Representación

### Diagramas de flujo

Definición: Representación gráfica de un algoritmo usando símbolos como flechas, rectángulos y rombos.

Ejemplo: Un diagrama para calcular el área de un triángulo.

### Pseudocódigo

Definición: Representación textual simple y cercana al lenguaje natural de un algoritmo.

Ejemplo:

```

Inicio

Leer base, altura

area <- (base \* altura) / 2

Imprimir area

Fin

```

## 3. Diseño y estilo de programación

### a. Relación entre algoritmo y programa

Un **algoritmo** es una serie de pasos bien definidos para resolver un problema específico, mientras que un **programa** es una implementación concreta de un algoritmo en un lenguaje de programación. La relación entre ambos es fundamental, ya que el algoritmo describe la lógica de la solución, y el programa es la materialización de esa lógica en un código que puede ser ejecutado por una computadora.

**Descripción de la Relación**

**1. Algoritmo**: Es una secuencia finita de instrucciones que describen cómo realizar una tarea o resolver un problema. Los algoritmos son independientes de cualquier lenguaje de programación y pueden ser representados en pseudocódigo, diagramas de flujo o simplemente como una lista de pasos.

**2. Programa**: Es una implementación del algoritmo en un lenguaje de programación específico (como Java, Python, C++, etc.). Un programa toma los pasos descritos en el algoritmo y los traduce a un formato que puede ser comprendido y ejecutado por una computadora. Los programas también pueden incluir la gestión de entradas y salidas, el manejo de errores y la interacción con otros programas o usuarios.

**Descripción de la Relación**

* **El algoritmo define qué hacer**: El algoritmo establece el conjunto de operaciones necesarias para resolver un problema de manera eficiente, sin preocuparse de cómo se van a realizar estas operaciones en términos de código.
* **El programa implementa cómo hacerlo**: Un programa toma el algoritmo y lo implementa usando un lenguaje de programación, añadiendo detalles como la sintaxis, estructuras de control (condiciones, bucles), manejo de memoria, y otros aspectos prácticos necesarios para que la solución sea ejecutada en una computadora.

Un **algoritmo bien diseñado** proporciona una **base sólida** sobre la cual se puede construir un **programa funcional y eficiente**. La calidad del algoritmo influye directamente en la eficiencia y efectividad del programa.

**Ejemplo de la Relación**

Supongamos que tienes el siguiente **algoritmo** para calcular la suma de los números del 1 al N:

1. Inicializa una variable suma en 0.
2. Para cada número i desde 1 hasta N:
   * Añadir i a suma.
3. Devolver el valor de suma.

Este algoritmo puede ser **traducido a un programa** en Java de la siguiente manera:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

**Conclusión**

La relación entre **algoritmo** y **programa** es que el primero es una representación abstracta y lógica de la solución, mientras que el segundo es la materialización de esa solución en código ejecutable. Un buen algoritmo es crucial para desarrollar un programa eficiente y funcional, ya que proporciona una guía clara sobre cómo resolver el problema en cuestión, y el programa es la forma en que esa solución se convierte en una herramienta utilizable por una computadora.

### b. Lineamientos y convenciones de programación

Los **lineamientos y convenciones de programación** son un conjunto de reglas y buenas prácticas que los programadores siguen para escribir código que sea claro, legible, mantenible y colaborativo. Estos estándares ayudan a que el código sea más comprensible tanto para el propio programador como para otros miembros del equipo, facilitando la colaboración y la resolución de errores.

**Importancia de los Lineamientos y Convenciones**

* **Legibilidad**: El código debe ser fácil de leer para entender su propósito y funcionamiento.
* **Mantenibilidad**: Facilita la actualización y modificación del código en el futuro.
* **Colaboración**: Un estándar común permite que múltiples desarrolladores trabajen en el mismo proyecto sin confusión.
* **Prevención de errores**: Las buenas prácticas ayudan a evitar errores comunes y a identificar problemas rápidamente.

**Ejemplos de Lineamientos y Convenciones**

1. **Usar nombres de variables significativos**:
   * **Descripción**: Las variables deben tener nombres descriptivos que reflejen su propósito o el tipo de información que almacenan.
   * **Ejemplo**:
   * Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

     Descripción generada automáticamente**Explicación**: Usar nombres como x, y, temp, o var1 hace que el código sea difícil de entender. Nombres descriptivos como totalVentas o nombreUsuario ayudan a identificar la función de la variable sin necesidad de leer el código que la rodea.
2. **Seguir estándares de indentación**:
   * **Descripción**: La indentación consiste en usar espacios o tabulaciones para alinear bloques de código de forma estructurada.
   * **Ejemplo**:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* + **Explicación**: La indentación clara hace que el código sea más fácil de leer y entender la jerarquía de los bloques de código. Los estándares de indentación comunes incluyen usar 4 espacios o 1 tabulación para cada nivel de anidación.

1. **Uso de comentarios**:
   * **Descripción**: Los comentarios deben usarse para explicar bloques de código complejos, describir el propósito de funciones o hacer anotaciones importantes.
   * **Ejemplo**:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* + **Explicación**: Los comentarios deben ser claros y concisos. No es necesario comentar líneas de código evidentes como int a = 10; a menos que su propósito no sea obvio en el contexto.

1. **Consistencia en el uso de mayúsculas y minúsculas**:
   * **Descripción**: Utilizar convenciones como camelCase para nombres de variables y PascalCase para nombres de clases es una práctica común.
   * **Ejemplo**:
   * Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

     Descripción generada automáticamente**Explicación**: camelCase (e.g., numeroDeEstudiantes) se utiliza para nombres de variables y métodos, mientras que PascalCase (e.g., CalculadoraDeImpuestos) se emplea para nombres de clases y estructuras.
2. **Evitar el uso de números mágicos**:
   * **Descripción**: Evitar el uso de valores numéricos arbitrarios y en su lugar, utilizar constantes con nombres descriptivos.
   * **Ejemplo**:
   * Interfaz de usuario gráfica, Texto, Aplicación

     Descripción generada automáticamente**Explicación**: El uso de nombres descriptivos para constantes ayuda a entender el propósito del valor y facilita su modificación en un solo lugar si es necesario.
3. **Uso de estructuras de control de manera clara**:
   * **Descripción**: Asegurarse de que las estructuras de control como if, for, y while estén bien organizadas y con bloques de código claros.
   * **Ejemplo**:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* + **Explicación**: Usar llaves {} incluso en bloques de una sola línea es una buena práctica para evitar errores y mejorar la legibilidad.

**Conclusión**

Los **lineamientos y convenciones de programación** son esenciales para garantizar un código limpio, organizado y fácil de entender. Usar nombres de variables significativos, seguir estándares de indentación, escribir comentarios útiles y mantener la consistencia en el estilo del código son algunas de las prácticas más importantes. Estos principios no solo mejoran la calidad del código, sino que también facilitan la colaboración entre desarrolladores y la capacidad de mantenimiento del proyecto a largo plazo.

## 4. Técnicas de comprobación

### a. Corrida de escritorio

Definición: Simulación manual del algoritmo para verificar su funcionamiento. La **corrida de escritorio** es una técnica utilizada para verificar manualmente el funcionamiento de un algoritmo. Consiste en simular paso a paso la ejecución del algoritmo en papel, siguiendo los pasos que describe el algoritmo y anotando los valores de las variables en cada etapa para asegurarse de que el algoritmo se comporte como se espera.

**Definición**

**Corrida de escritorio**: Es un método de análisis y verificación en el que se ejecuta un algoritmo de manera manual, anotando los valores de las variables y los resultados intermedios en cada paso para validar su lógica y asegurarse de que se obtiene el resultado correcto.

**Importancia de la Corrida de Escritorio**

* **Verificación de lógica**: Permite detectar errores en la lógica del algoritmo antes de implementarlo en un lenguaje de programación.
* **Comprensión del flujo**: Ayuda a comprender cómo se desarrollan los pasos de un algoritmo y cómo interactúan entre sí las variables.
* **Facilita la depuración**: Identificar posibles errores en el algoritmo en una etapa temprana facilita la corrección antes de escribir el código.

**Ejemplo de Corrida de Escritorio**

Imaginemos un algoritmo simple para calcular la suma de los números enteros desde 1 hasta N (por ejemplo, N = 4).

**Algoritmo**:

1. Inicializar suma a 0.
2. Para i desde 1 hasta N (inclusive):
   * Añadir i a suma.
3. Devolver suma.

**Corrida de escritorio** para N = 4:

* Inicializar suma = 0.
* Iteración 1 (i = 1):
  + suma = suma + i → suma = 0 + 1 = 1.
* Iteración 2 (i = 2):
  + suma = suma + i → suma = 1 + 2 = 3.
* Iteración 3 (i = 3):
  + suma = suma + i → suma = 3 + 3 = 6.
* Iteración 4 (i = 4):
  + suma = suma + i → suma = 6 + 4 = 10.

**Resultado final**: suma = 10.

**Pasos para Realizar una Corrida de Escritorio**

1. **Escribir el algoritmo**: Describir los pasos del algoritmo de manera detallada, ya sea en pseudocódigo o en un diagrama de flujo.
2. **Elegir un caso de prueba**: Seleccionar un conjunto de datos de entrada para simular la ejecución del algoritmo.
3. **Seguir los pasos manualmente**: Ejecutar cada paso del algoritmo en papel, actualizando los valores de las variables y anotando los resultados intermedios.
4. **Verificar el resultado**: Comparar el resultado obtenido con el resultado esperado para confirmar si el algoritmo es correcto.
5. **Detectar errores**: Si el resultado no es el esperado, identificar en qué paso se produjo el error y corregir la lógica del algoritmo.

**Ejemplo de Corrida de Escritorio con un Algoritmo de Mayoría**

Supongamos que queremos verificar un algoritmo que determina si un número dado es mayor que la mayoría de los elementos de una lista. Consideremos la lista [2, 5, 1, 7, 3] y el número 4.

**Algoritmo**:

1. Inicializar mayoria a false.
2. Contar cuántos elementos en la lista son mayores que 4.
3. Si la cantidad de elementos mayores que 4 es mayor que la mitad del tamaño de la lista, asignar mayoria a true.
4. Devolver mayoria.

**Corrida de escritorio**:

* Lista: [2, 5, 1, 7, 3], Número 4.
* Inicializar mayoria = false.
* Contar elementos mayores que 4:
  + Elementos mayores que 4: 5, 7.
* La cantidad de elementos mayores es 2.
* La mitad del tamaño de la lista es 5 / 2 = 2.5.
* Como 2 < 2.5, mayoria permanece false.

**Resultado final**: mayoria = false.

**Conclusión**

La **corrida de escritorio** es una técnica útil para validar la lógica de un algoritmo antes de implementarlo en código. Ayuda a identificar errores en la lógica y permite asegurar que el algoritmo se comporta como se espera con ejemplos concretos. Esta práctica es esencial en el desarrollo de algoritmos, ya que proporciona una manera de verificar la precisión de la solución y facilita la depuración antes de escribir el código.

### b. Inducción matemática

La **inducción matemática** es un método de demostración utilizado para probar que una afirmación es verdadera para todos los números enteros positivos, o para una secuencia de elementos. Este método es fundamental en el análisis de algoritmos y la teoría de la computación, ya que permite demostrar formalmente la validez de ciertos algoritmos y propiedades matemáticas de manera rigurosa.

**Definición**

**Inducción matemática**: Es una técnica de prueba que se utiliza para establecer que una afirmación es verdadera para todos los elementos de un conjunto infinito (generalmente, los números naturales). Se basa en dos pasos principales: la base de inducción y el paso inductivo.

**Pasos de la Inducción Matemática**

1. **Base de inducción**:
   * Demostrar que la afirmación es cierta para el caso más pequeño del conjunto (por ejemplo, n = 1).
   * Este paso establece que la afirmación se cumple para el primer valor de n.
2. **Paso inductivo**:
   * Suponer que la afirmación es cierta para un valor arbitrario k, y luego demostrar que esto implica que la afirmación es cierta para k + 1.
   * Esta suposición se llama **hipótesis de inducción**.
3. **Conclusión**:
   * Una vez que se han completado ambos pasos, se concluye que la afirmación es verdadera para todos los números enteros mayores o iguales a n (en general, para todos los n en el dominio considerado).

**Ejemplo de Inducción Matemática**

Supongamos que queremos probar la siguiente afirmación:

**Afirmación**: La suma de los primeros n números enteros positivos es igual a n(n + 1) / 2.

**Fórmula a demostrar**: 1+2+3+⋯+n=n(n+1)21 + 2 + 3 + \dots + n = \frac{n(n + 1)}{2}1+2+3+⋯+n=2n(n+1)​.

**Paso 1: Base de inducción**

* Para n = 1, la fórmula es: 1=1(1+1)2=22=1.1 = \frac{1(1 + 1)}{2} = \frac{2}{2} = 1.1=21(1+1)​=22​=1.
* La afirmación es cierta para n = 1.

**Paso 2: Paso inductivo**

**Tabla

Descripción generada automáticamente**

**Paso 3: Conclusión**

* Hemos demostrado que si la afirmación es cierta para n = k, entonces también es cierta para n = k + 1. Por lo tanto, por el principio de inducción matemática, la afirmación es verdadera para todos los números enteros positivos n.

**Aplicación de la Inducción Matemática en Algoritmos**

La **inducción matemática** es especialmente útil para probar propiedades de algoritmos recursivos o de estructuras de datos, como en el caso de:

* **Complejidad de algoritmos**: Demostrar que un algoritmo tiene una complejidad de tiempo o espacio específica.
* **Correctitud de algoritmos**: Verificar que un algoritmo produce los resultados correctos para todas las entradas posibles.
* **Propiedades de estructuras de datos**: Probar que una estructura de datos como un árbol binario o una lista enlazada mantiene ciertas propiedades a lo largo de sus operaciones.

**Ejemplo de Inducción en Algoritmos**

Supongamos que queremos probar que un algoritmo de suma recursiva para calcular la suma de los primeros n números naturales es correcto. La función recursiva es:

java

Copiar código

public int sumaRecursiva(int n) {

if (n == 1) {

return 1;

} else {

return n + sumaRecursiva(n - 1);

}

}

**Prueba de correctitud**:

**Texto

Descripción generada automáticamenteConclusión**

La **inducción matemática** es una herramienta poderosa para demostrar la validez de algoritmos y propiedades matemáticas. A través de la base de inducción y el paso inductivo, se puede probar que una afirmación es verdadera para todos los números en un conjunto específico. Este método no solo es fundamental en la teoría de algoritmos, sino que también proporciona una forma rigurosa de garantizar la correcta implementación y análisis de algoritmos y estructuras de datos.

### c. Depuración y pruebas

Definición: Proceso de identificar y corregir errores en el código.

### d. Tipos de errores

Definición y ejemplos:

- Sintácticos: Falta de un punto y coma en C.

- Semánticos: Usar una fórmula incorrecta.

- De ejecución: División por cero.

# III. Lógica de Programación

## 1. Estructuras y tipos de datos

### Definición y clasificación:

En Java, los tipos de datos se dividen en:

* Tipos primitivos: Son los tipos de datos básicos que no se derivan de otros. Ejemplos: int, double, char, boolean.
* Tipos de datos de referencia: Son tipos que se refieren a objetos o instancias de clases. Ejemplos: String, Array, List.

Ejemplo:

java

Copiar código

int numero = 5; // tipo primitivo (entero)

double temperatura = 23.5; // tipo primitivo (decimal)

char letra = 'A'; // tipo primitivo (carácter)

boolean esVerdadero = true; // tipo primitivo (booleano)

String saludo = "Hola"; // tipo de dato de referencia (cadena de texto)

## 2. Literales, variables y constantes

* Literales: Son valores literales que se usan directamente en el código, como 10, 3.14, 'A', "Texto".
* Variables: Son espacios de almacenamiento cuyo valor puede cambiar durante la ejecución del programa.
* Constantes: Se declaran con la palabra clave final y su valor no puede cambiar después de la asignación.

Ejemplo:

java

Copiar código

final int MAX\_USUARIOS = 100; // constante

int edad = 30; // variable

## 3. Tipos de operadores y expresiones

De asignación:

* Se utilizan para asignar valores a las variables. Ejemplos: =, +=, -=, \*=.

java

Copiar código

int a = 5; // Asignación simple

a += 3; // a ahora es 8

Aritméticos:

* Operadores que realizan operaciones matemáticas. Ejemplos: +, -, \*, /, %.

java

Copiar código

int suma = 3 + 2; // 5

int division = 10 / 2; // 5

int resto = 10 % 3; // 1

Relacionales:

* Comparan dos valores y devuelven true o false. Ejemplos: ==, !=, <, >, <=, >=.

java

Copiar código

boolean esIgual = (5 == 5); // true

boolean esMayor = (10 > 5); // true

Lógicos:

* Operadores que combinan expresiones booleanas. Ejemplos: && (Y lógico), || (O lógico), ! (NO lógico).

java

Copiar código

boolean resultado = (5 > 3) && (2 < 4); // true

boolean resultadoNegado = !(5 == 5); // false

Jerarquía de operadores: La jerarquía de operadores define el orden de evaluación de las operaciones. Los operadores aritméticos tienen mayor precedencia que los de asignación, y los operadores de comparación tienen mayor precedencia que los lógicos.

## 4. Elaboración de un programa

Ejemplo de programa en Java: Programa que solicita un número al usuario y muestra si es par o impar.

java

Copiar código

import java.util.Scanner;

public class ParImpar {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Introduce un número: ");

int numero = scanner.nextInt();

if (numero % 2 == 0) {

System.out.println("El número es par.");

} else {

System.out.println("El número es impar.");

}

scanner.close();

}

}

# IV. Elementos Básicos de Programación

## 1. Construcciones básicas

Secuencia: Las instrucciones se ejecutan una tras otra en orden.

Decisión simple, doble y múltiple:

* Simple:

java

Copiar código

if (condición) {

// ejecutar si la condición es verdadera

}

* Doble:

java

Copiar código

if (condición) {

// ejecutar si la condición es verdadera

} else {

// ejecutar si la condición es falsa

}

* Múltiple:

java

Copiar código

int opcion = 2;

switch (opcion) {

case 1:

System.out.println("Opción 1 seleccionada");

break;

case 2:

System.out.println("Opción 2 seleccionada");

break;

default:

System.out.println("Opción no válida");

}

Iteraciones:

* Bucle for:

java

Copiar código

for (int i = 0; i < 5; i++) {

System.out.println("Iteración " + i);

}

* Bucle while:

java

Copiar código

int i = 0;

while (i < 5) {

System.out.println("Iteración " + i);

i++;

}

## 2. Herramientas de programación

Contadores y acumuladores:

* Contadores: Variables que se incrementan para contar la cantidad de veces que ocurre un evento.

java

Copiar código

int contador = 0;

for (int i = 0; i < 10; i++) {

contador++;

}

* Acumuladores: Variables que acumulan un total.

java

Copiar código

int suma = 0;

for (int i = 1; i <= 5; i++) {

suma += i; // suma los números del 1 al 5

}

Banderas y centinelas:

* Banderas: Variables booleanas que indican si se ha cumplido una condición.

java

Copiar código

boolean encontrado = false;

if (/\* condición \*/) {

encontrado = true;

}

* Centinelas: Valores especiales que indican el fin de una estructura de datos, como -1 en un array.

Estados: Representan el estado actual de una condición o proceso, como activo o inactivo.

## 3. Deducción de series

Ejemplo de suma de los primeros n números naturales:

java

Copiar código

int suma = 0;

int n = 5; // número de términos

for (int i = 1; i <= n; i++) {

suma += i;

}

System.out.println("La suma de los primeros " + n + " números naturales es: " + suma);

## 4. Desarrollo de algoritmos básicos

Conversión de unidades: Conversión de metros a kilómetros.

java

Copiar código

double metros = 5000;

double kilometros = metros / 1000;

System.out.println(metros + " metros son " + kilometros + " kilómetros.");

Cálculo de promedios:

java

Copiar código

int suma = 0;

int cantidad = 5;

Scanner scanner = new Scanner(System.in);

for (int i = 0; i < cantidad; i++) {

System.out.print("Introduce un número: ");

int numero = scanner.nextInt();

suma += numero;

}

double promedio = (double) suma / cantidad;

System.out.println("El promedio es: " + promedio);

Ecuaciones de segundo grado:

java

Copiar código

import java.lang.Math;

double a = 1, b = -3, c = 2;

double discriminante = b \* b - 4 \* a \* c;

if (discriminante >= 0) {

double raiz1 = (-b + Math.sqrt(discriminante)) / (2 \* a);

double raiz2 = (-b - Math.sqrt(discriminante)) / (2 \* a);

System.out.println("Raíces: " + raiz1 + " y " + raiz2);

} else {

System.out.println("No hay raíces reales.");

}

Ecuaciones simultáneas: Usar métodos de sustitución o eliminación para resolver sistemas de ecuaciones lineales.

## 5. Algoritmos clásicos

Algoritmo de Euclides (MCD):

java

Copiar código

int mcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

Factorial no recursivo:

java

Copiar código

int factorial = 1;

int n = 5;

for (int i = 1; i <= n; i++) {

factorial \*= i;

}

System.out.println("El factorial de " + n + " es: " + factorial);

Serie de Fibonacci no recursiva:

java

Copiar código

int a = 0, b = 1, temp, n = 10;

System.out.print("Serie de Fibonacci: ");

for (int i = 0; i < n; i++) {

System.out.print(a + " ");

temp = a + b;

a = b;

b = temp;

}

System.out.println();

Detección de números primos:

java

Copiar código

boolean esPrimo(int num) {

if (num < 2) return false;

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) return false;

}

return true;

}

# V. Estructuras de Datos Estáticas

## 1. Clasificación de estructuras estáticas

En Java, las estructuras estáticas son aquellas que tienen un tamaño definido en tiempo de compilación, como arreglos.

## 2. Vectores: definición y operaciones básicas

Definición: Los vectores en Java son arrays que permiten almacenar múltiples elementos del mismo tipo.

Ejemplo:

java

Copiar código

int[] numeros = new int[5]; // vector de 5 elementos

numeros[0] = 10;

numeros[1] = 20;

Operaciones básicas:

java

Copiar código

for (int i = 0; i < numeros.length; i++) {

System.out.println("Elemento en la posición " + i + ": " + numeros[i]);

}

## 3. Matrices: definición y operaciones básicas

Definición: Una matriz es un array bidimensional.

Ejemplo:

java

Copiar código

int[][] matriz = new int[3][3]; // matriz de 3x3

matriz[0][0] = 1;

matriz[1][1] = 5;

matriz[2][2] = 9;

Operaciones básicas:

java

Copiar código

for (int i = 0; i < matriz.length; i++) {

for (int j = 0; j < matriz[i].length; j++) {

System.out.print(matriz[i][j] + " ");

}

System.out.println();

}

## 4. Cadenas

Definición: Las cadenas en Java se representan con la clase String.

Operaciones básicas:

java

Copiar código

String saludo = "Hola, mundo!";

System.out.println("Longitud: " + saludo.length());

System.out.println("Mayúsculas: " + saludo.toUpperCase());

System.out.println("Subcadena: " + saludo.substring(0, 5));

Manipulación:

java

Copiar código

String frase = "Java es divertido";

String[] palabras = frase.split(" "); // divide la cadena por espacios

for (String palabra : palabras) {

System.out.println(palabra);

}

# VI. Funciones y Procedimientos

## 1. Conceptos generales

En Java, las funciones se definen como métodos dentro de una clase y pueden devolver un valor o ser de tipo void si no devuelven nada.

Ejemplo:

java

Copiar código

public int sumar(int a, int b) {

return a + b;

}

## 2. Paso de parámetros: por valor

En Java, los parámetros se pasan por valor, lo que significa que se pasa una copia del valor de la variable.

Ejemplo:

java

Copiar código

public void cambiarValor(int x) {

x = 10;

}

## 3. Modularidad

Es el proceso de dividir el programa en módulos (métodos o funciones) para mejorar la organización y reutilización del código.

Ejemplo:

java

Copiar código

public void mostrarMensaje() {

System.out.println("¡Hola desde un método modular!");

}

## 4. Funciones que reciban arreglos

Ejemplo de función que recibe un arreglo:

java

Copiar código

public void imprimirArreglo(int[] arr) {

for (int num : arr) {

System.out.println(num);

}

}

Ejercicios

**1)** Declara dos variables numéricas (con el valor que desees), muestra por consola la suma, resta, multiplicación, división y módulo (resto de la división).

**2)** Declara 2 variables numéricas (con el valor que desees), he indica cual es mayor de los dos. Si son iguales indicarlo también. Ves cambiando los valores para comprobar que funciona.

**3)** Declara un String que contenga tu nombre, después muestra un mensaje de bienvenida por consola. Por ejemplo: si introduzco **«Fernando»**, me aparezca **«Bienvenido Fernando».**

**4)** Modifica la aplicación anterior, para que nos pida el nombre que queremos introducir (recuerda usar JOptionPane).

**5)** Haz una aplicación que calcule el área de un círculo(pi\*R2). El radio se pedirá por teclado (recuerda pasar de String a double con **Double.parseDouble**). Usa la constante PI y el método pow de Math.

**6)**Lee un número por teclado e indica **si** es divisible entre 2 (resto = 0). Si no lo es, también debemos indicarlo.

**7)** Lee un número por teclado y muestra por consola, el carácter al que pertenece en la tabla ASCII. Por ejemplo: si introduzco un 97, me muestre una **a.**

**8)** Modifica el ejercicio anterior, para que en lugar de pedir un número, pida un carácter (char) y muestre su código en la tabla ASCII.

**9)** Lee un número por teclado que pida el precio de un producto (puede tener decimales) y calcule el precio final con IVA. El IVA sera una constante que sera del 21%.

**10)** Muestra los números del 1 al 100 (ambos incluidos). Usa un bucle while.

**11)** Haz el mismo ejercicio anterior con un bucle for.

**12)** Muestra los números del 1 al 100 (ambos incluidos) divisibles entre 2 y 3. Utiliza el bucle que desees.

**13)** Realiza una aplicación que nos pida un número de ventas a introducir, después nos pedirá tantas ventas por teclado como número de ventas se hayan indicado. Al final mostrara la suma de todas las ventas. Piensa que es lo que se repite y lo que no.

**14)** Realiza una aplicación que nos calcule una ecuación de segundo grado. Debes pedir las variables **a, b** y **c** por teclado y comprobar antes que el **discriminante** (operación en la raíz cuadrada). Para la raíz cuadrada usa el método **sqlrt** de **Math.**Te recomiendo que uses mensajes de traza.

**15)** Lee un número por teclado y comprueba que este numero es mayor o igual que cero, si no lo es lo volverá a pedir (do while), después muestra ese número por consola.

**16)** Escribe una aplicación con un String que contenga una contraseña cualquiera. Después se te pedirá que introduzcas la contraseña, con 3 intentos. Cuando aciertes ya no pedirá mas la contraseña y mostrara un mensaje diciendo **«Enhorabuena»**. Piensa bien en la condición de salida (3 intentos y si acierta sale, aunque le queden intentos).

**17)** Crea una aplicación que nos pida un día de la semana y que nos diga si es un dia laboral o no. Usa un **switch** para ello.

**18)** Pide por teclado dos número y genera 10 números aleatorios entre esos números. Usa el método **Math.random** para generar un número entero aleatorio (recuerda el casting de double a int).

**19)** Pide por teclado un número entero positivo (debemos controlarlo) y muestra  el número de cifras que tiene. Por ejemplo: si introducimos 1250, nos muestre que tiene 4 cifras. Tendremos que controlar si tiene una o mas cifras, al mostrar el mensaje.

**28)**Eliminar los espacios de una frase pasada por consola por el usuario.

**29)** Pedir al usuario que nos escriba frases de forma infinita hasta que insertemos una cadena vacia. Mostrar la cadena resultante

**31)** Mostrar la longitud de una cadena.

**32)** Pedir dos palabras por teclado, indicar si son iguales.

**33)** Dada una cadena, extraer la cuarta y quinta letra usando el método substring.

Métodos de la clase String de java

La clase **String** en Java proporciona una amplia variedad de métodos para manipular y procesar cadenas de texto. Aquí tienes una lista de los métodos más importantes, organizados por su funcionalidad:

**Creación y Características**

1. **length()**  
   Devuelve la longitud de la cadena.

java

Copiar código

String str = "Hola";

int longitud = str.length(); // 4

1. **isEmpty()**  
   Verifica si la cadena está vacía.

java

Copiar código

String str = "";

boolean vacia = str.isEmpty(); // true

1. **isBlank()** *(Java 11)*  
   Verifica si la cadena está vacía o contiene solo espacios en blanco.

java

Copiar código

String str = " ";

boolean blanco = str.isBlank(); // true

**Comparación**

1. **equals(String otro)**  
   Compara dos cadenas considerando mayúsculas y minúsculas.

java

Copiar código

"Hola".equals("hola"); // false

1. **equalsIgnoreCase(String otro)**  
   Compara dos cadenas ignorando mayúsculas y minúsculas.

java

Copiar código

"Hola".equalsIgnoreCase("hola"); // true

1. **compareTo(String otro)**  
   Compara dos cadenas lexicográficamente.

java

Copiar código

"Hola".compareTo("hola"); // resultado negativo

1. **compareToIgnoreCase(String otro)**  
   Compara dos cadenas ignorando mayúsculas y minúsculas.

java

Copiar código

"Hola".compareToIgnoreCase("hola"); // 0

**Búsqueda**

1. **contains(CharSequence secuencia)**  
   Verifica si la cadena contiene una secuencia de caracteres.

java

Copiar código

"Hola mundo".contains("mundo"); // true

1. **startsWith(String prefijo)**  
   Verifica si la cadena comienza con un prefijo.

java

Copiar código

"Hola mundo".startsWith("Hola"); // true

1. **endsWith(String sufijo)**  
   Verifica si la cadena termina con un sufijo.

java

Copiar código

"Hola mundo".endsWith("mundo"); // true

1. **indexOf(String subcadena)**  
   Devuelve la posición de la primera aparición de una subcadena.

java

Copiar código

"Hola mundo".indexOf("mundo"); // 5

1. **lastIndexOf(String subcadena)**  
   Devuelve la posición de la última aparición de una subcadena.

java

Copiar código

"Hola mundo mundo".lastIndexOf("mundo"); // 11

**Transformación**

1. **toLowerCase()**  
   Convierte la cadena a minúsculas.

java

Copiar código

"Hola".toLowerCase(); // "hola"

1. **toUpperCase()**  
   Convierte la cadena a mayúsculas.

java

Copiar código

"Hola".toUpperCase(); // "HOLA"

1. **trim()**  
   Elimina los espacios en blanco al principio y al final.

java

Copiar código

" Hola ".trim(); // "Hola"

1. **strip()** *(Java 11)*  
   Similar a trim(), pero elimina cualquier espacio Unicode.

java

Copiar código

" Hola ".strip(); // "Hola"

1. **repeat(int n)** *(Java 11)*  
   Repite la cadena n veces.

java

Copiar código

"Hola".repeat(3); // "HolaHolaHola"

**Extracción y Subcadenas**

1. **charAt(int index)**  
   Devuelve el carácter en una posición específica.

java

Copiar código

"Hola".charAt(1); // 'o'

1. **substring(int beginIndex)**  
   Devuelve una subcadena desde el índice especificado.

java

Copiar código

"Hola mundo".substring(5); // "mundo"

1. **substring(int beginIndex, int endIndex)**  
   Devuelve una subcadena desde el índice inicial hasta el índice final (sin incluirlo).

java

Copiar código

"Hola mundo".substring(0, 4); // "Hola"

**Reemplazo y Divisiones**

1. **replace(char antiguo, char nuevo)**  
   Reemplaza todas las ocurrencias de un carácter.

java

Copiar código

"Hola".replace('a', 'o'); // "Holo"

1. **replace(CharSequence antigua, CharSequence nueva)**  
   Reemplaza todas las ocurrencias de una subcadena.

java

Copiar código

"Hola mundo".replace("mundo", "Java"); // "Hola Java"

1. **replaceAll(String regex, String reemplazo)**  
   Reemplaza las coincidencias de una expresión regular.

java

Copiar código

"123-456".replaceAll("\\d", "X"); // "XXX-XXX"

1. **split(String regex)**  
   Divide la cadena en un arreglo de cadenas según un delimitador.

java

Copiar código

"Hola,mundo".split(","); // ["Hola", "mundo"]

**Conversiones**

1. **valueOf(Tipo valor)**  
   Convierte diferentes tipos a cadenas.

java

Copiar código

String numero = String.valueOf(123); // "123"

1. **toCharArray()**  
   Convierte la cadena en un arreglo de caracteres.

java

Copiar código

char[] caracteres = "Hola".toCharArray(); // ['H', 'o', 'l', 'a']

1. **join(CharSequence delimitador, CharSequence... elementos)** *(Java 8)*  
   Une elementos con un delimitador.

java

Copiar código

String unidos = String.join("-", "Hola", "mundo"); // "Hola-mundo"

Estos métodos hacen de la clase **String** una herramienta muy poderosa para trabajar con cadenas de texto en Java. ¡Espero que te sea útil!