Universidad Nacional Experimental De Guayana
Vicerrectorado Académico
Coordinación General De Pregrado
Proyecto De Carrera: Ingeniería Informática
Lenguajes Y Compiladores
Semestre VI
Sección 1

Los lenguajes de programación.

Profesor:
Felix Marquez
Participantes:
Anyelis Coro ci:30.366.262
Antoni Medina ci:30.857.435
William Figuera ci:24.412.167

Ciudad Guayana, Mayo del 2025.

Introducción

Los lenguajes de programación son sistemas estructurados de comunicación que permiten a los humanos dar instrucciones a las computadoras para realizar tareas específicas. Estos lenguajes actúan como intermediarios entre el pensamiento lógico humano y el código binario (ceros y unos) que las máquinas pueden procesar. Su evolución ha sido clave en el desarrollo de la informática, permitiendo la creación de software, aplicaciones web, sistemas operativos, inteligencia artificial y más.

En el contexto actual, su estudio trasciende la mera escritura de código: implican comprender modelos de memoria, gestión de recursos, y patrones de diseño, siendo herramientas esenciales para implementar soluciones en campos como:

Computación distribuida (Go, Rust).
Análisis de datos a escala (Python, R).
Verificación formal de software (Ada, SPARK).

Desarrollo

1) Explique los diferentes paradigmas de lenguajes de programación.

- Paradigma Imperativo

  Definición: Se enfoca en describir paso a paso cómo resolver un problema mediante comandos que modifican el estado del programa.

  Características: Usa variables, asignaciones, bucles (como for y while) y funciones.

  Ejemplos: C, Pascal, BASIC.

  Código de ejemplo en C:

  ```
  int suma = 0;
  for (int i = 1; i <= 10; i++) {
  suma += i;
  }
  ```

- Paradigma Orientado a Objetos (POO)

  Definición: Organiza el código en "objetos" que contienen datos (atributos) y comportamientos (métodos).

  Conceptos clave: Clases, herencia, polimorfismo y encapsulamiento.

  Ejemplos: Java, C++, Python.

  Código de ejemplo en Java:

  ```
  class Persona {
  private String nombre;
  public Persona(String n) { this.nombre = n; }
  ```

```
 public void saludar() { System.out.println("Hola, soy " + nombre); }
}
```

- Paradigma Funcional

  Definición: Trata la programación como la evaluación de funciones matemáticas, evitando cambios de estado y datos mutables.

  Características: Funciones puras, recursión, y operaciones como map y filter.

  Ejemplos: Haskell, Lisp, Erlang.

  Código de ejemplo en Haskell:

```
sumaLista :: [Int] -> Int
sumaLista [] = 0
sumaLista (x:xs) = x + sumaLista xs
```

- Paradigma Lógico

  Definición: Basado en lógica formal y reglas para resolver problemas.

  Características: Usa hechos y reglas para deducir soluciones.

  Ejemplo principal: Prolog.

  Código de ejemplo en Prolog:

```
padre(juan, maria).
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
```

- Paradigma Multiparadigma

  Definición: Combina varios paradigmas en un solo lenguaje.

  Ejemplos: Python (POO + funcional), JavaScript (imperativo + POO + funcional).

Código de ejemplo en Python:

```
numeros = [1, 2, 3]
cuadrados = list(map(lambda x: x**2, numeros))
```

Comparación Rápida:

- ➔ Imperativo: Ideal para control detallado (ej: sistemas operativos).
- ➔ POO: Bueno para proyectos grandes y modulares (ej: aplicaciones empresariales).
- ➔ Funcional: Excelente para procesamiento de datos y concurrencia.
- ➔ Lógico: Útil para IA y sistemas basados en reglas.

2) Para un autómata dado, implementarlo en diferentes lenguajes, presente una tabla comparativa con la ejecución en diferentes escenarios, concluya cuál de los lenguajes resolvió el problema más rápido. El autómata se explica a continuación: considere la creación de una lista, cada nodo contiene, partida, cuerpo y firma digital. El arranque como es natural inicia con el primero nodo, luego los próximos nodos uno detrás del otro simulando una lista enlazada simple, está constituida por n nodos y, k elementos aleatorios en el cuerpo de cada nodo, donde el valor de k es el mismo para todos los nodos.

En detalle:

- **partida:** a) El primer nodo de la lista tiene una firma digital sha256 del día y hora de la creación, b) para los próximos nodos es la copia del valor firma digital del nodo que le precede.
- **cuerpo:** para todos los nodos contiene k elementos de enteros aleatorios entre 1 y 100.000.
- **firma digital:** para todos los nodos este valor representa el valor sha256 del contenido de partida concatenado con los elementos del cuerpo separados por un espacio.

Escenarios para la ejecución con los lenguajes seleccionados a) n=3 y k=4, b) n=10 y k=200, c) n=200 y k=10. En la figura se muestran los datos alfanuméricos de ejemplo que debe tener el cuerpo, estos se deben generar de forma aleatoria…

Códigos:

Python

```
import hashlib
import random
from datetime import datetime
class Nodo:
    def __init__(self, partida, cuerpo):
        self.partida = partida
        self.cuerpo = cuerpo
self.firma = hashlib.sha256((partida + " " + " ".join(map(str, cuerpo))).encode()).hexdigest()
def generar_lista(n, k):
```

```
    lista = []
    fecha = datetime.now().strftime("%d/%m/%Y %H:%M")
    partida_actual = hashlib.sha256(fecha.encode()).hexdigest()
    for _ in range(n):
        cuerpo = [random.randint(1, 100000) for _ in range(k)]
        nodo = Nodo(partida_actual, cuerpo)
        lista.append(nodo)
        partida_actual = nodo.firma
    return lista

# Ejemplo para n=3, k=4
lista = generar_lista(3, 4)
for nodo in lista: print(f"Partida: {nodo.partida[:64]}\nCuerpo: {nodo.cuerpo}\nFirma:
{nodo.firma[:64]}\n")
```

C++

```
#include <iostream>
#include <vector>
#include <ctime>
#include <sstream>
#include <iomanip>
#include <cstdint>
#include <string>
#include <algorithm>
```

```
#include <chrono>
#include <iomanip>
#include "sha.h"
struct Nodo
{
    std::string partida;
    std::vector<int> cuerpo;
    std::string firma;
    Nodo(std::string p, std::vector<int> c) : partida(p), cuerpo(c)
    {
        std::string concat = partida + " ";
        for (int num : cuerpo)
            concat += std::to_string(num) + " ";
        firma = sha256(concat);
    }
};
struct ResultadoCaso
{
    int n;
    int k;
    double tiempo_ms;
};
std::vector<Nodo> generar_lista(int n, int k, double &tiempo_ms)
{
    auto inicio = std::chrono::high_resolution_clock::now();
```

```
  std::vector<Nodo> lista;
    time_t now = time(0);
    std::string partida_actual = sha256(std::ctime(&now));
    for (int i = 0; i < n; i++)
    {
        std::vector<int> cuerpo;
        for (int j = 0; j < k; j++)
            cuerpo.push_back(rand() % 100000 + 1);
        lista.emplace_back(partida_actual, cuerpo);
        partida_actual = lista.back().firma;
    }
    auto fin = std::chrono::high_resolution_clock::now();
            tiempo_ms    =    std::chrono::duration_cast<std::chrono::milliseconds>(fin    -
inicio).count();
    return lista;
}
void mostrar_tabla_comparativa(const std::vector<ResultadoCaso> &resultados)
{
    std::cout << "\n=== TABLA COMPARATIVA DE TIEMPOS ===\n";
    std::cout << "+------+-------+------------+\n";
    std::cout << "|   n  |   k   | Tiempo (ms)|\n";
    std::cout << "+------+-------+------------+\n";
    for (const auto &res : resultados)
    {
        std::cout << "| " << std::setw(4) << res.n << " | "
```

```
                << std::setw(5) << res.k << " | "
                    << std::setw(10) << std::fixed << std::setprecision(2) << res.tiempo_ms
<< " |\n";
    }
    std::cout << "+------+-------+------------+\n";
}
int main()
{
    std::vector<ResultadoCaso> resultados;
    std::vector<std::pair<int, int>> casos = {
        {3, 4},    // Caso 1
        {10, 200}, // Caso 2
        {200, 10}  // Caso 3
    };
    srand(time(0));
    for (const auto &caso : casos)
    {
        double tiempo = 0.0;
        auto lista = generar_lista(caso.first, caso.second, tiempo);
        resultados.push_back({caso.first, caso.second, tiempo});
        // Mostrar detalles del primer nodo como ejemplo
        if (!lista.empty())
        {
            std::cout << "\nCaso n=" << caso.first << ", k=" << caso.second << " - Primer
nodo:\n";
```

```
            std::cout << "Partida: " << lista[0].partida.substr(0, 64) << "...\n";
            std::cout << "Cuerpo (" << lista[0].cuerpo.size() << " elementos): ";
            for (size_t j = 0; j < std::min(5, (int)lista[0].cuerpo.size()); j++)
                std::cout << lista[0].cuerpo[j] << " ";
            if (lista[0].cuerpo.size() > 5)
                std::cout << "...";
            std::cout << "\nFirma: " << lista[0].firma.substr(0, 64) << "...\n";
        }
    }
    mostrar_tabla_comparativa(resultados);
    return 0;
}
```

Java

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
class Nodo {
    String partida;
    List<Integer> cuerpo;
    String firmaDigital;
    public Nodo(String partida, List<Integer> cuerpo, String firmaDigital) {
        this.partida = partida;
```

```
        this.cuerpo = cuerpo;
        this.firmaDigital = firmaDigital;
    }
    @Override
    public String toString() {
        return "Nodo{\n" +
                "  partida='" + partida + "'\n" +
                "  cuerpo=" + cuerpo + "\n" +
                "  firmaDigital='" + firmaDigital + "'\n" +
                '}';
    }
}
public class prueba {
    private static final Random random = new Random();
    private static MessageDigest digest;
    static {
        try {
            digest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
    public static void main(String[] args) {
        // Escenarios a probar
        int[][] escenarios = { { 3, 4 }, { 10, 200 }, { 200, 10 } };
```

```
    for (int[] escenario : escenarios) {
        int n = escenario[0];
        int k = escenario[1];
        System.out.println("\nEjecutando escenario: n=" + n + ", k=" + k);
        long inicio = System.currentTimeMillis();
        List<Nodo> lista = crearListaNodos(n, k);
        long fin = System.currentTimeMillis();
        long tiempoEjecucion = fin - inicio;
        System.out.println("Tiempo de ejecución: " + tiempoEjecucion + " ms");
        // Mostrar los primeros 3 nodos para verificación
        if (lista.size() > 0) {
            System.out.println("\nPrimer nodo:");
            System.out.println(lista.get(0));
            if (lista.size() > 1) {
                System.out.println("\nSegundo nodo:");
                System.out.println(lista.get(1));
                if (lista.size() > 2) {
                    System.out.println("\nTercer nodo:");
                    System.out.println(lista.get(2));
                }
            }
        }
    }
}
```

```
public static List<Nodo> crearListaNodos(int n, int k) {
    List<Nodo> lista = new ArrayList<>();
    String partidaAnterior = null;
    for (int i = 0; i < n; i++) {
        // Generar cuerpo con k elementos aleatorios
        List<Integer> cuerpo = generarCuerpo(k);
        // Determinar partida
        String partida;
        if (i == 0) {
            partida = generarPartidaInicial();
        } else {
            partida = partidaAnterior;
        }
        // Generar firma digital
        String firmaDigital = generarFirmaDigital(partida, cuerpo);
        // Crear nodo y añadir a la lista
        Nodo nodo = new Nodo(partida, cuerpo, firmaDigital);
        lista.add(nodo);
        // Actualizar partida anterior para el próximo nodo
        partidaAnterior = firmaDigital;
    }
    return lista;
}
private static List<Integer> generarCuerpo(int k) {
    List<Integer> cuerpo = new ArrayList<>();
```

```
        for (int i = 0; i < k; i++) {
            cuerpo.add(random.nextInt(100000) + 1); // Números entre 1 y 100000
        }
        return cuerpo;
    }
    private static String generarPartidaInicial() {
        String fechaHora = java.time.LocalDateTime.now().toString();
        return calcularSHA256(fechaHora);
    }
    private static String generarFirmaDigital(String partida, List<Integer> cuerpo) {
        StringBuilder sb = new StringBuilder(partida);
        for (Integer num : cuerpo) {
            sb.append(" ").append(num);
        }
        return calcularSHA256(sb.toString());
    }
        private static String calcularSHA256(String input) {            byte[] hash =
digest.digest(input.getBytes(java.nio.charset.StandardCharsets.UTF_8));
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1)
                hexString.append('0');
            hexString.append(hex);
        }
```

```
        return hexString.toString();
    }
}
```

Tabla Comparativa de Rendimiento

Lenguaje	n=3,k=4**(ms)**	n=10,k=200**(ms)**	n=200,k=10**(ms)**
Python	0.36	1.04	1.69
C++	0.0	7.0	13.0
Java	23.0	5.0	21.0

3) Presente la estructura morfológica (léxica y sintáctica) de los lenguajes (C++, rust).

Léxico:

- C ++: Se refiere al vocabulario básico del lenguaje, formado por elementos que el compilador reconoce y utiliza para entender el código. Estos elementos, incluyen palabras reservadas, identificadores, constantes,

operadores, signos de puntuación y otros elementos como tipos de datos y literales. Como int, float, if, while, class, etc.

- Rust: Palabras reservadas(fin, let, mut, match, impl, trait, struct, etc), conceptos de ownership (move, borrow, &, &mut), macros (println!, vec!, macros definidas por el usuario), tipos especiales (Option<T>, Result<T , E>, tipos genericos), Lifetimes (a, static, etc)

Sintaxis:

- C++

  Declaración de clases:

  class MiClase {

  public:

  int mi_metodo();

  private:

  int mi_variable;

  };

  Herencia:

  class Derivada : public Base {};

  Plantilla:

  template <typename T>

  T max(T a, T b) { return a > b ? a : b; }

- Rust:

  Declaración de estructuras:

```
struct MiStruct {
    campo: i32,
}
```

Implementación de traits:

```
impl MiTrait for MiStruct {}
```

Match expressions:

```
match valor {
    Some(x) => println!("{}", x),
    None => println!("Nada"),
}
```

4) Plantee un problema A el cual requiere de una interfaz hombre máquina donde el usuario dará órdenes al computador (lenguaje alto nivel), presente una propuesta de lenguaje de programación L, que permita interactuar al usuario con la máquina mediante un programa, en otras palabras, el propósito del lenguaje es permitir al usuario dar respuesta al problema A. Es necesario que el lenguaje incorpore estructura de control y bucles, escriba dos ejemplos de programas con su lenguaje para resolver un problema en particular.

Ejemplo:

Se cuenta con un sistema de suministro de agua, contiene: 1) dos tanques de agua, 2)sensores de nivel del tanque, 3) sensor presión de agua que llega a las bombas de llenado, 4) sensor de temperatura de cada motor, 5) sensor medidor de caudal de salida de la bomba. El sistema contiene un driver el cual permite tanto leer como

accionar cada elemento del sistema. Se creará un lenguaje que permite a un operador dar los comandos necesarios o crear programas que permiten crear tareas automatizadas. Se definirán palabras claves, para el lenguaje L: inicio, encender_motor(id),obeterTemperaturaMotor(id),caudalSalidaMotor(id), presiónentradaMotor(id) si (condicion) entonces, mientras(condicion)...

Luego de esa presentación del lenguaje L para esa planta o tanques de agua, presente unos programas escrito en L y que resuelva un planteamiento en particular.

Problema A:

Automatizar el sistema de riego en un invernadero equipado con:

- 2 depósitos de agua (Tanque A y Tanque B).
- Sensores de nivel (0-100%), humedad del suelo (3 zonas), temperatura ambiental y lluvia.
- Válvulas de riego controlables por zona.

Objetivo del Lenguaje L:

Permitir a operadores sin experiencia en programación:

- Crear reglas de riego basadas en condiciones ambientales.
- Monitorear y ajustar el sistema en tiempo real.

2. Especificación del Lenguaje L

Palabras Clave y Sintaxis

Comando	**Descripción**	**Ejemplo**

inicio	Inicia el programa.	inicio
apagar_valvula(id)	Cierra la válvula de una zona específica.	apagar_valvula("Zona1")
abrir_valvula(id)	Abre la válvula de una zona.	abrir_valvula("Zona 2")
nivel_tanque(id)	Devuelve el % de llenado del tanque (ej: "TanqueA").	si (nivel_tanque("TanqueA") > 20)
humedad_suelo(id)	Devuelve la humedad de la zona (ej: "Zona3").	humedad_suelo("Zona 3") < 30
temperatura_ambiente()	Devuelve la temperatura en °C.	si (temperatura_ambiente() > 15)
esta_lloviendo()	Devuelve True o False.	si (no esta_lloviendo())

si (condición) entonces	Ejecuta acciones si se cumple la condición.	Ver Ejemplo 1
mientras (condición)	Repite acciones mientras sea verdadera.	Ver Ejemplo 2
esperar(minutos)	Pausa el programa.	esperar(60)
fin	Termina el programa.	fin

Ejemplos de Programas en Lenguaje L

- Ejemplo 1: Riego por Humedad

  Objetivo: Regar "Zona1" solo si su humedad es <30% y hay agua en "TanqueA".

```
inicio
  si (humedad_suelo("Zona1") < 30 y nivel_tanque("TanqueA") > 20) entonces
    abrir_valvula("Zona1")
    esperar(10)              # Riega por 10 minutos
    apagar_valvula("Zona1")
  fin_si
fin
```

❖ Ejemplo 2: Riego Nocturno Automatizado

Objetivo: Regar todas las zonas entre 2:00 AM y 5:00 AM, solo si no llueve.

```
inicio
  mientras (True) hacer          # Bucle infinito
    si (temperatura_ambiente() > 15 y no esta_lloviendo()) entonces
      abrir_valvula("Zona1")
      abrir_valvula("Zona2")
      esperar(15)              # Riego por 15 minutos
       apagar_valvula("Zona1")
       apagar_valvula("Zona2")
    fin_si
    esperar(60)                # Verifica cada hora
  fin_mientras
fin
```

Ventajas y Extensiones Futuras

Ventajas:

- Intuitivo: Sintaxis similar a lenguaje natural.
- Seguro: Evita riego con tanques vacíos o durante lluvia.
- Eficiente: Reduce el consumo de agua y energía.

Extensiones Propuestas:

1. Alertas:

   enviar_alerta("TanqueA vacío") # Notifica al operador.

2. Programación Horaria:

   a_las("14:00", [abrir_valvula("Zona3")]) # Acción a hora fija.

Conclusión

Los lenguajes de programación son herramientas fundamentales que han transformado la interacción humano-máquina, permitiendo traducir lógica abstracta en instrucciones ejecutables.

Los lenguajes de programación son pilares de la innovación tecnológica. Su selección depende del contexto: desde sistemas críticos que requieren eficiencia (C++, Rust) hasta aplicaciones rápidas en entornos dinámicos (Python, JavaScript). La tendencia apunta hacia lenguajes más seguros, expresivos y especializados, democratizando el desarrollo mientras enfrentan desafíos como la computación cuántica y la sostenibilidad energética.

En esencia, son el puente entre ideas humanas y soluciones digitales, impulsando progreso en todos los campos del conocimiento.

Referencias Bibliográficas (Normas APA)

Libros

- Aho, A. V., Lam, M. S., Sethi, R. & Ullman, J. D. (2007). Compilers: Principles, techniques, and tools (2a ed.). Pearson. Incluye fundamentos teóricos de diseño de lenguajes y compiladores.
- Sebesta, R. W. (2019). Concepts of programming languages (12a ed.). Pearson. Aborda comparación de paradigmas y evolución histórica.
- Stroustrup, B. (2013). The C++ programming language (4a ed.). Addison-Wesley. Ejemplo de eficiencia en sistemas críticos (C++).

Artículos Académicos

- Hughes, J. (1989). Why functional programming matters. The Computer Journal, *32*(2), 98-107. https://doi.org/10.1093/comjnl/32.2.98

Tendencias en lenguajes funcionales.

- Matsakis, N. D. & Klock, F. S. (2014). The Rust language. ACM SIGPLAN Notices, *49*(10), 1-1. https://doi.org/10.1145/2692956.2663188
- Seguridad en lenguajes modernos (Rust).