Informe Escrito: Análisis y Justificación del Sistema de Control de Velocidad de Escobas

1.1 Análisis del problema

Descripción del problema

Mi código tiene como objetivo verificar si una escoba voladora excede o no un límite de velocidad predeterminado. Para hacer esto, comparo la velocidad actual de la escoba (que se ingresa en metros por segundo) con el límite de velocidad que se establezca (que se proporciona en kilómetros por hora).

Como las unidades son diferentes, también tuve que convertir la velocidad de metros por segundo a kilómetros por hora para que la comparación sea precisa y tenga sentido.

Requisitos funcionales

- El sistema debe aceptar como entrada la velocidad de la escoba en metros por segundo.
- El sistema debe aceptar como entrada el límite de velocidad en kilómetros por hora.
- El sistema debe convertir la velocidad de la escoba a m/s a km/h.
- El sistema debe comparar la velocidad convertida con el límite establecido.
- El sistema debe informar si la escoba excede el límite y por cuántos km/h.
- El sistema debe informar si la escoba está dentro del límite y cuánto margen queda.

Requisitos no funcionales

- El sistema debe ser fácil de usar, con mensajes claros para el usuario.
- El sistema debe manejar correctamente los tipos de datos (enteros y decimales).
- El sistema debe mostrar los resultados con un formato legible y comprensible.
- La solución debe funcionar tanto en C++ como en Python.

Casos de uso principales

Caso de uso: Verifique si la escoba excede el límite de velocidad

Actor principal: Usuario (probablemente un inspector de tránsito mágico)

Precondición: Ninguna

Flujo básico:

El usuario ingresa la velocidad de la escoba en m/s

El usuario ingresa el límite de velocidad en km/h

El sistema convierte la velocidad de m/s a km/h

El sistema compara la velocidad convertida con el límite.

El sistema muestra si la escoba excede o no el límite, junto con la diferencia

Identificación de entradas, procesos y salidas

Entradas:

1. Velocidad de la escoba en metros por segundo (tipo entero o decimal)
2. Límite de velocidad en kilómetros por hora (tipo entero o decimal)

Procesos:

1. Conversión de unidades (m/sa km/h) multiplicando por 3.6
2. Comparación de la velocidad convertida con el límite
3. Cálculo de la diferencia entre la velocidad y el límite

Salidas:

1. Mensaje indicando si la escoba excede el límite de velocidad
2. Valor numérico que indica la diferencia (exceso o margen) en km/h

1.2 Justificación de la solución

Estrategia elegida

Para resolver este problema, decidí usar un enfoque modular.Osea enfocarme en dividir el problema en partes más pequeñas para hacerlo más organizado y facil. Básicamente, creé una función para comparar las velocidades; en C++ la llamé compararVelocidadEscoba, y en Python, comparar_velocidad_escoba. Esta forma de organizar el código no solo lo hace más claro y fácil de entender, sino que también me ayudó a separar la lógica principal (los

cálculos) de la parte que interactúa con el usuario (mostrar los resultados). Siempre trato de estructurar mi código de esta manera porque me ayuda a mantenerlo ordenado y fácil de modificar en el futuro.

La solución sigue estos pasos:

Obtener datos del usuario a través de la entrada estándar.

Convierte la velocidad de m/s a km/h mediante la multiplicación por 3.6.

Realice la comparación lógica usando una estructura condicional simple.

Devuelva un mensaje explicativo que incluya el resultado de la comparación.

Justificación de estructuras de datos y algoritmos.

Tipos de datos simples: Se utilizan dos diferentes tipos de datos (enteros y decimales) para manejar los valores numéricos, lo cual es suficiente para este problema.

Estructura condicional if-else: Permite partir en dos el flujo del programa según el resultado de la comparación. Esta estructura es ideal para este problema binario (excede/no excede).

Funciones con parámetros y retorno: Permiten encapsular la lógica de conversión y comparación, facilitando la modularidad y reutilización del código.

Cadenas para formatear: Se utilizan para generar mensajes de salida claros y comprensibles que incluyen los valores calculados.

Comparación con soluciones alternativas

Solución actual (función dedicada):

Ventajas: Clara separación de responsabilidades, código modular y reutilizable.

Desventajas: Podría ser más rápido y específico pero no es muy importante

Alternativa: Todo el código junto:

Ventajas: Podría ser un poco más rápido

Desventajas: Más desordenado y difícil de leer

Alternativa: Usar clases (programación orientada a objetos):

Ventajas: Más organizado para problemas complejos

Desventajas: Demasiado complicado para algo tan simple

La solución actual la elegí por su equilibrio entre ser un código simple y bien hecho. El usar funciones proporciona modularidad sin agregar complejidad innecesaria, lo que hace que el código sea fácil de entender, hacer y mantener. Además, esta estructura facilita futuras mejoras del sistema, como agregar más tipos de vehículos mágicos o diferentes límites según zonas.

1.3 Diagrama UML