CS2023 – Programación III

Práctica Calificada #1a

Rubén Rivas Medina

Marzo 2025

Indicaciones Específicas

- **Tiempo límite:** 50 minutos
- **Formato de entrega:**
  - La solución debe implementarse en un archivo cabecera (‘.h‘) y su correspondiente implementación (‘.cpp‘):
    - geodesy.h (Declaración de la clase)
    - geodesy.cpp (Implementación de métodos)
  - Los archivos deben nombrarse exactamente como se indica (sensibles a mayúsculas)
- **Plataforma de entrega:**
  - Subir directamente los archivos a **https://www.gradescope.com** ó
  - Se puede crear un archivo comprimido solucion.zip que contenga o subirlos directamente:
    - geodesy.h, geodesy.cpp

Restricciones Clave

- **Prohibido usar** std::vector, std::unique_ptr u otros contenedores STL
- Debe gestionarse memoria manualmente con new[] y delete[]

Recomendaciones a tomar en cuenta que podrian afectar su calificación

- **Asegurarse que cada nuevo metodo o atributo que agrega funcione.** a lo más podria fallar el ultimo método u operador implementado pero no todos, comentando el ultimo el resto funciona.
- **Asegurese de seguir las convenciones que se le solicitan**, nombres de clases, namespace, metodos exactamente igual a los solicitados en las pruebas, un uso incorrecto es sospecha del uso de un Chatbox.
- **Si la clase es totalmente inoperativa** (requiere cambios estructurales mayores para funcionar) **o contiene errores graves** (memory leaks críticos, crashes, implementación incorrecta de funcionalidad básica), La calificación máxima podria ser reducida hasta un **60%** del valor total de lo calificado.

Pregunta unica

En un proyecto de simulación astronómica, necesitas calcular distancias entre observatorios basados en mediciones angulares del Sol. Inspirado en el método de **Eratóstenes** (que midió la Tierra usando ángulos y distancias), debes implementar una clase GeoMeasurement que:

$$\text{Circunferencia} = \left(\frac{360}{|\theta_1 - \theta_2|} \right) \times \text{Distancia} \tag{1}$$

Donde:

- θ_1 y θ_2 son ángulos registrados en dos ubicaciones
- **Distancia** es la separación lineal en kilómetros entre ellas

Esqueleto de la Clase

```cpp
#include <iostream>
#include <string>

// ...

class GeoMeasurement {
private:
    double* angles;      // Array dinámico de ángulos
    size_t num_angles;   // Número actual de ángulos
    double distance;     // Distancia en kilómetros
    std::string label;   // Nombre de la medición

public:
    // --- Constructores ---

    // --- Métodos ---

    // --- Operadores ---

    };
```

Instrucciones

1. Implementar todos los constructores y destructor
2. Método add_angle debe redimensionar el array dinámicamente
3. Sobrecargar operadores +, = (copia y movimiento) y «
4. calculate_circumference retorna -1.0 en errores

NOTA: Manejo de Múltiples Ángulos para el calculo de Circunferencia

1. Cuando una instancia de GeoMeasurement contiene **más de 2 ángulos**:
    - La clase debe permitir almacenar N ángulos (solo limitado por memoria disponible)
    - Solo utilizar **el primero y el ultimo de los ángulos** para el cálculo
    - Los demás ángulos se ignoran (pero se mantienen almacenados)

Ejemplos de Prueba

Test 1: Constructor básico y añadir ángulos

Prueba de funcionalidad básica

```
// Crear medición básica y añadir ángulos
double angulos_iniciales[] = {0.12};
geodesy::GeoMeasurement medicion1(angulos_iniciales, 1, 800, "Base");
medicion1.add_angle(0.13);
medicion1.add_angle(0.14);

std::cout << medicion1 << "\n";
// Salida esperada: "Base: 3 angulos, 800 km"

// Verificar cálculo
double circ = medicion1.calculate_circumference();
if (circ < 0) {
    std::cerr << "Error: datos insuficientes\n";
} else {
    std::cout << "Circunferencia calculada: " << circ << " km\n";
}
```

Test 2: Constructor de copia

Prueba de copia profunda

```
// Crear original y copia
double angs[] = {0.15, 0.16};
geodesy::GeoMeasurement original(angs, 2, 500, "Original");
geodesy::GeoMeasurement copia = original;

// Modificar copia
copia.add_angle(0.17);

std::cout << "Original: " << original << "\n";
// Salida: "Original: 2 angulos, 500 km"
std::cout << "Copia: " << copia << "\n";
// Salida: "Copia: 3 angulos, 500 km"
```

Test 3: Constructor de movimiento

Prueba de semántica de movimiento

```
geodesy::GeoMeasurement temporal({0.18, 0.19}, 300, "Temporal");

// Robar recursos del objeto temporal
geodesy::GeoMeasurement movido = std::move(temporal);

std::cout << "Movido: " << movido << "\n";
// Salida: "Movido: Temporal: 2 angulos, 300 km"
std::cout << "Temporal: " << temporal << "\n";
// Salida: "Temporal: 0 angulos, 0 km" (estado válido pero vacío)
```

Test 4: Operadores y manejo de errores

Prueba de combinación y errores

```
// Crear dos mediciones y combinarlas
geodesy::GeoMeasurement m1({0.20}, 200, "M1");
geodesy::GeoMeasurement m2({0.21, 0.22}, 300, "M2");
geodesy::GeoMeasurement combinada = m1 + m2;

std::cout << combinada << "\n";
// Salida: "M1 + M2: 3 angulos, 500 km"

// Intentar cálculo con datos insuficientes
double resultado = combinada.calculate_circumference();
if (resultado < 0) {
    std::cerr << "Error calculo: necesita exactamente 2 angulos\n";
    // Salida esperada: este mensaje de error
}
```

Rúbrica de Evaluación

Criterio	Ponderación	Descripción
Correctitud	60 %	▪ Manejo adecuado de memoria dinámica (no leaks, liberación correcta) ▪ Implementación precisa de constructores de copia/movimiento ▪ Cálculos geodésicos exactos según fórmula especificada ▪ Sobrecarga correcta de operadores
Robustez	30 %	▪ Manejo adecuado de casos límite (arrays vacíos, divisiones por cero) ▪ Validación de parámetros en todos los métodos ▪ Uso apropiado de retornos de error (no se exigue el uso de excepciones)
Legibilidad	10 %	▪ Estilo de código consistente (indentación, nombrado) ▪ Comentarios explicativos donde sea necesario ▪ Organización lógica de métodos ▪ Encabezados claros en archivos