CS2023 - Programación III

**Fundamentos de Programación - Tarea #1**

Rubén Rivas Medina

Marzo 2025

## Ejercicio 1: vector_3d

### Descripción

Implementar la clase vector_3d en el namespace geometry::3d. La clase:

- Contiene tres componentes: x, y y z.
- Incluye constructores para inicializar el vector.
- Sobrecarga los operadores aritméticos (+ y -) y el operador de asignación.
- Declara una función friend para calcular el producto punto (dot product).

### Unit Tests Sugeridos

Unit Tests para vector_3d

```cpp
#include <cassert>
#include "vector_3d.h"

void test_vector_initialization() {
    geometry::vector_3d v(1, 2, 3);
    assert(v.get_x() == 1 && v.get_y() == 2 && v.get_z() == 3);
}

void test_vector_addition() {
    geometry::vector_3d v1(1, 2, 3);
    geometry::vector_3d v2(4, -2, 0);
    geometry::vector_3d sum = v1 + v2;
    assert(sum.get_x() == 5 && sum.get_y() == 0 && sum.get_z() == 3);
}

void test_vector_dot_product() {
    geometry::vector_3d v1(1, 2, 3);
    geometry::vector_3d v2(4, -2, 0);
    double dot = geometry::3d::dot_product(v1, v2); // 1*4 + 2*(-2) + 3*0 = 0
    assert(dot == 0);
}
```

## Ejercicio 2: polynomial

### Descripción

Implementar la clase polynomial en el namespace math::operations. Esta clase:

- Representa un polinomio mediante un arreglo dinámico de coeficientes.
- Incluye constructores (parametrizado y de copia) y operador de asignación para gestionar la memoria.
- Sobrecargar el operador + para sumar polinomios el operador - para restar polinomios y el operador * para multiplicarlos.
- Sobrecargar el operador « para imprimir el polinomio.
- Declara una función friend, por ejemplo, evaluate, que reciba un polinomio y un valor $x$ para evaluarlo.

### Unit Tests Sugeridos

Unit Tests para polynomial

```cpp
#include <cassert>
#include "polynomial.h"

void test_polynomial_initialization() {
    double coeffs[] = {2, 3, 4}; // Representa 2 + 3x + 4x^2
    math::operations::polynomial poly(2, coeffs);
    assert(poly.get_coefficient(0) == 2);
    assert(poly.get_coefficient(1) == 3);
    assert(poly.get_coefficient(2) == 4);
}

void test_polynomial_addition() {
    double coeffs1[] = {1, 2}; // 1 + 2x
    double coeffs2[] = {3, 4}; // 3 + 4x
    math::operations::polynomial poly1(1, coeffs1);
    math::operations::polynomial poly2(1, coeffs2);
    math::operations::polynomial sum = poly1 + poly2;
    assert(sum.get_coefficient(0) == 4);
    assert(sum.get_coefficient(1) == 6);
}

void test_polynomial_evaluation() {
    double coeffs[] = {1, 2, 1}; // 1 + 2x + x^2
    math::operations::polynomial poly(2, coeffs);
    double result = math::operations::evaluate(poly, 3); // 1 + 6 + 9 = 16
    assert(result == 16);
}

void test_polynomial_producto() {
    double cs1[] = {2, 3, 4}; // Representa 2 + 3x + 4x^2
    const math::operations::polynomial p1(2, cs1);
    CHECK(p1.get_coefficient(0) == 2);
    CHECK(p1.get_coefficient(1) == 3);
    CHECK(p1.get_coefficient(2) == 4);
    double cs2[] = {4, 5, 2}; // Representa 4 + 5x + 2x^2
    const math::operations::polynomial p2(2, cs2);
    const auto p3 = p1 * p2;
    std::cout << p3 << std::endl;   // 8 22x^1 35x^2 26x^3 8x^4
    double cs3[] = {10, 2}; // Representa 10 + 2x
    const math::operations::polynomial p4(1, cs3);
    const auto p5 = p1 * p4;
    std::cout << p5 << std::endl;   // 20 34x^1 46x^2 8x^3
}
```

## Ejercicio 3: point

### Descripción

Desarrollar la clase point en el namespace geometry. La clase:

- Almacena dos coordenadas: x e y.
- Incluye constructores para inicialización.
- Sobrecargar los operadores +, - y, opcionalmente, == para comparar puntos.
- Sobrecargar el operador « para imprimir el point.

### Unit Tests Sugeridos

Unit Tests para point

```cpp
#include <cassert>
#include "point.h"

void test_point_initialization() {
    geometry::point p(3, 4);
    assert(p.get_x() == 3 && p.get_y() == 4);
}

void test_point_addition() {
    geometry::point p1(3, 4);
    geometry::point p2(1, 2);
    geometry::point p3 = p1 + p2;
    assert(p3.get_x() == 4 && p3.get_y() == 6);
}

void test_point_equality() {
    geometry::point p1(3, 4);
    geometry::point p2(3, 4);
    assert(p1 == p2);
}

void test_point_print() {
    const geometry::point p1(3, 4);
    const geometry::point p2(3, 4);
    const auto p3 = p1 + p2;
    std::cout << p3 << std::endl;
}
```

### Ejercicio 4: logger

#### Descripción

Implementar la clase logger en el namespace file::operations. La clase:

- Abre un archivo en el constructor y lo cierra en el destructor (RAII).
- Sobrecarga el operador « para insertar mensajes en el log.
- Ofrece métodos para escribir mensajes.

#### Unit Tests Sugeridos

Unit Tests para logger

```
#include <cassert>
#include <fstream>
#include "logger.h"

void test_file_logger_write() {
    file::operations::logger flogger("temp_test.txt");
    flogger.write("Test message");
    std::ifstream infile("temp_test.txt");
    std::string line;
    std::getline(infile, line);
    assert(line.find("Test message") != std::string::npos);
    infile.close();
}

void test_file_logger_operator_insertion() {
    file::operations::logger flogger("temp_test.txt");
    flogger << "Another message";
    std::ifstream infile("temp_test.txt");
    std::string line;
    std::getline(infile, line);
    assert(line.find("Another message") != std::string::npos);
    infile.close();
}

void test_file_logger_multiple_writes() {
    file::operations::logger flogger("temp_test.txt");
    flogger.write("First");
    flogger.write("Second");
    std::ifstream infile("temp_test.txt");
    std::string line;
    int count = 0;
    while (std::getline(infile, line)) {
        if (line.find("First") != std::string::npos ||
            line.find("Second") != std::string::npos) {
                ++count;
            }
    }
    assert(count >= 2);
    infile.close();
}
```

### Ejercicio 5: system_log

#### Descripción

Implementar la clase system_log en el namespace performance::monitoring. La clase:

- Mide y registra métricas de rendimiento (por ejemplo, tiempos de ejecución de secciones de código) en un archivo de salida.
- Posee un constructor que configura la ruta del archivo de salida.
- Provee métodos para iniciar y detener cronómetros asociados a tareas específicas, utilizando la biblioteca estándar <chrono>.
- Registra las métricas en un archivo colocando el mensaje de inicio (start) y luego el mensaje de fin (stop) separados por comas, permitiendo un análisis posterior del rendimiento.

#### Unit Tests Sugeridos

Unit Tests para system_log

```cpp
#include <cassert>
#include <fstream>
#include "system_log.h"
// Test para medir una única tarea.
void test_performance_monitor_single_measure() {
    performance::monitoring::system_log pm("perf_log.csv");
    pm.start("task1:start");
    // Simular una tarea (por ejemplo, un retardo)
    pm.stop("task1:stop");
    std::ifstream infile("perf_log.csv");
    std::string line;
    std::getline(infile, line);
    assert(line == "task1:start,task1:stop");
    infile.close();
}
// Test para medir múltiples tareas.
void test_performance_monitor_multiple_measures() {
    performance::monitoring::system_log pm("perf_log.csv");
    pm.start("task1:start");
    // Simular tarea 1
    pm.stop("task1:stop");
    pm.start("task2:start");
    // Simular tarea 2
    pm.stop("task2:stop");
    std::ifstream infile("perf_log.csv");
    int count = 0;
    std::string line;
    while (std::getline(infile, line)) {
        ++count;
    }
    assert(count == 2);
    infile.close();
}
// Test para verificar el formato de salida.
void test_performance_monitor_format() {
    performance::monitoring::system_log pm("perf_log.csv");
    pm.start("start_log");
    // Simular tarea
    pm.stop("stop_log");
    std::ifstream infile("perf_log.csv");
    std::string line;
    std::getline(infile, line);
    // Verificar que el formato incluya comas
    assert(line == "start_log,stop_log");
    infile.close();
}
```

### Ejercicio 6: product

#### Descripción

Implementar la clase product en el namespace company. En esta alternativa:

- Se define la clase product que contiene atributos como name (std::string), price (double) y weight (double).
- Se incluyen constructores para inicialización.
- Se sobrecargan los operadores == y < para comparar productos (por ejemplo, basándose en precio y peso).
- Se declara una función friend, compare_by_value, que compara productos según su ratio precio/peso.

#### Unit Tests Sugeridos

Unit Tests para product

```cpp
#include <cassert>
#include "product.h"

void test_product_initialization() {
    company::product prod("gadget", 29.99, 1.5);
    assert(prod.get_name() == "gadget");
    assert(prod.get_price() == 29.99);
    assert(prod.get_weight() == 1.5);
}

void test_product_equality() {
    company::product prod1("widget", 19.99, 0.8);
    company::product prod2("widget", 19.99, 0.8);
    company::product prod3("widget", 19.99, 1.0);
    assert(prod1 == prod2);
    assert(!(prod1 == prod3));
}

void test_product_compare_by_value() {
    company::product prod1("item_a", 10.0, 2.0); // Ratio = 5.0
    company::product prod2("item_b", 15.0, 5.0); // Ratio = 3.0
    bool result = company::compare_by_value(prod2, prod1);
    assert(result == true);
}
```

### Ejercicio 7: fraction

#### Descripción

Desarrollar la clase fraction en el namespace math::operations. La clase:

- Almacena un numerador y un denominador.
- Incluye constructores (por defecto y parametrizado), sobrecarga de operadores aritméticos y de igualdad.
- Declara una función friend para simplificar o comparar fracciones.

#### Unit Tests Sugeridos

Unit Tests para fraction

```
#include <cassert>
#include "fraction.h"

void test_fraction_simplification() {
    math::operations::fraction f1(2, 4);
    math::operations::fraction f2(1, 2);
    assert(f1 == f2);
}

void test_fraction_arithmetic() {
    math::operations::fraction f1(1, 2);
    math::operations::fraction f2(1, 2);
    math::operations::fraction sum = f1 + f2;
    assert(sum.get_numerator() == 1 && sum.get_denominator() == 1);
}

void test_fraction_friend_function() {
    math::operations::fraction f1(3, 9);
    math::operations::fraction expected(1, 3);
    assert(f1 == expected);
}
```

### Ejercicio 8: bag

#### Descripción

Implementar la clase bag en el namespace structures. Esta estructura se utiliza para almacenar elementos sin necesidad de un orden específico y solo admite inserción y recorrido (iteración). Se requiere:

- Almacenar internamente los elementos (por ejemplo, mediante una lista enlazada simple).
- Incluir un constructor por defecto y, si se desea, un constructor que permita inicializar la colección.
- Un método add() para insertar elementos.
- Un método begin() para ubicarse en el primer valor, retorna el puntero al primer valor.
- Un método end() para ubicar el utimo valor del bag, normalmente retorna nullptr.

#### Unit Tests Sugeridos

Unit tests para bag

```cpp
#include <cassert>
#include "bag.h"

void test_bag_insertion() {
    structures::bag b;
    b.add(10);
    b.add(20);
    // Se asume que bag provee un método size() o similar.
    assert(b.size() == 2);
}

void test_bag_iteration() {
    structures::bag b;
    b.add(5);
    b.add(15);
    int sum = 0;
    for (auto ptr = b.begin(); ptr != b.end(); ++ptr) {
        sum += *ptr;
    }
    assert(sum == 20);
}

void test_bag_empty() {
    structures::bag b;
    // Se espera que bag esté vacío al inicio.
    assert(b.size() == 0);
}
```