Assignment1

> 如果网页公式渲染有误，请阅读本文件夹下的 `README.pdf`

设计并实现一个支持 n 维向量运算的简单库。该库应不依赖面向对象编程（即不使用类或对象），而是通过结构体和函数来实现向量的基本运算。

什么是 n 维向量？

向量是表示多维空间中一个点的位置的数学工具。在日常生活中，最常见的向量是二维和三维向量，分别用于描述平面和空间中的点。然而，在科学计算和工程应用中，向量的维数可以扩展到任意多个维度。我们称其为 **n 维向量**，其中 `n` 是向量的维度。

- 一个二维向量可以表示为 `(x, y)`，例如，二维平面中的点。
- 一个三维向量可以表示为 `(x, y, z)`，例如，空间中的点。
- 对于 n 维向量，可以表示为 `(x1, x2, x3, ..., xn)`，表示在 n 维空间中的一个位置。

n 维向量的常见运算包括向量加法、向量减法、点积、模长计算、归一化和叉积。这些操作在不同的数学和物理应用中有广泛的应用，例如机器学习中的高维数据处理、物理模拟中的多维矢量场等。

向量运算的介绍

1. 向量加法

向量加法是两个向量的对应分量相加，结果也是一个向量。例如，给定两个二维向量 `(x1, y1)` 和 `(x2, y2)`，它们的和为 `(x1 + x2, y1 + y2)`。对于 n 维向量，这个运算规则类似：

$$\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$$

2. 向量减法

向量减法是两个向量的对应分量相减。例如，二维向量 `(x1, y1)` 和 `(x2, y2)` 的差为 `(x1 - x2, y1 - y2)`。n 维向量的减法类似：

$$\mathbf{a} - \mathbf{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$$

3. 向量点积

向量点积是两个向量的对应分量相乘并求和，结果是一个标量。二维向量 `(x1, y1)` 和 `(x2, y2)` 的点积为 `x1 * x2 + y1 * y2`。n 维向量的点积计算公式为：

$$\mathbf{a} \cdot \mathbf{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + \cdots + a_n \cdot b_n$$

4. 向量模长

向量的模长（或称为向量的长度）是从原点到向量终点的距离，通常用欧几里得距离来计算。二维向量 `(x, y)` 的模长为 `sqrt(x^2 + y^2)` 。n 维向量的模长计算公式为：

$$|\mathbf{a}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

5. 向量归一化

归一化是将一个向量缩放为单位长度（即模长为 1），保留其方向。归一化的向量通常用于表示方向或单位矢量。对于向量 `a` ，归一化的计算公式为：

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|}$$

6. 向量叉乘（仅适用于三维向量）

向量叉乘仅适用于三维向量，它返回一个垂直于两个输入向量的向量。例如，给定向量 `a = (x1, y1, z1)` 和 `b = (x2, y2, z2)` ，它们的叉乘为：

$$\mathbf{a} \times \mathbf{b} = (y1 \cdot z2 - z1 \cdot y2, z1 \cdot x2 - x1 \cdot z2, x1 \cdot y2 - y1 \cdot x2)$$

叉乘在 n 维空间中没有广泛应用，但它在三维空间中用于计算法向量、扭矩等物理量。

7. 向量拷贝

为了确保在操作向量时不改变原始向量内容，向量库应支持拷贝操作。拷贝操作创建一个新向量，其所有分量与原向量相同。向量拷贝适用于需要对原向量进行多次操作，且不希望修改原数据的情况。

设计要求

1. 数据结构

- 定义一个结构体 `Vector` 来表示 n 维向量。结构体中包含一个指向 `float` 的指针来存储向量的各个分量，另有一个 `size_t` 类型的变量记录向量的维数。

```
struct {
    size_t dimension;  // 向量的维数
    float* data;       // 指向存储向量分量的动态数组
} Vector;
```

2. 基本操作函数

- 定义用于对 n 维向量进行操作的函数，具体要求如下：
  i. **向量初始化**:

- 编写一个函数 `create_vector`，接受向量的维数 `n`，返回一个 n 维零向量。

ii. **向量销毁**:
- 编写一个函数 `destroy_vector`，释放向量的内存空间。

iii. **向量加法**:
- 编写一个函数 `add`，接受两个向量并返回它们的和。若维数不匹配，返回错误。

iv. **向量减法**:
- 编写一个函数 `subtract`，接受两个向量并返回它们的差。若维数不匹配，返回错误。

v. **向量点积**:
- 编写一个函数 `dot_product`，计算两个 n 维向量的点积。若维数不匹配，返回错误。

vi. **向量模长**:
- 编写一个函数 `magnitude`，计算 n 维向量的模长（即向量的欧几里得长度）。

vii. **向量归一化**:
- 编写一个函数 `normalize`，对 n 维向量进行归一化处理，使其模长为 1。如果向量长度为 0，返回错误标志。

viii. **向量叉乘**（仅适用于三维向量）：
- 编写一个函数 `cross_product`，计算两个三维向量的叉乘。如果输入的向量维数不是 3，返回错误。

ix. **向量拷贝**:
- 编写一个函数 `copy_vector`，将一个向量的内容拷贝到新的向量中，返回一个与原向量内容相同的新向量。

3. 异常处理

- 所有操作必须检测向量的维度是否匹配，如维度不匹配，应返回错误值或进行适当的错误处理。
- 对零向量的归一化操作应该返回一个错误标志。

4. 函数接口要求

- 要求实现如下的接口

```
Vector create_vector(size_t n);
void destroy_vector(Vector& v);
Vector add(const Vector& a, const Vector& b);
Vector subtract(const Vector& a, const Vector& b);
float dot_product(const Vector& a, const Vector& b);
float magnitude(const Vector& a);
int normalize(Vector& a);  // 修改传入的向量，返回状态值
Vector cross_product(const Vector& a, const Vector& b);  // 三维向量叉乘
Vector copy_vector(const Vector& a);  // 向量拷贝
```

5. 测试用例

- 不提供测试用例，建议自行编写一组测试用例，验证向量加法、减法、点积、模长、归一化、叉乘和拷贝函数，确保它们在不同维度的向量上正确运行。

6. 内存管理

- 动态分配和释放内存必须正确，以防止内存泄漏。每个向量的 `data` 数组应在初始化时动态分配，并在销毁时释放。

7. 提交方式

- 发送到邮件到 submit@vollate.top，标题为 `ARTINX2025视觉Assignment1`
- 代码以附件提交，需要压缩为一个 zip 文件，附件名为 `姓名-学号-Assignment1.zip` ，自行替换为自己的姓名学号

> 务必按照以上格式提交，否则可能正常接收