

Практическая работа №7

В контексте выполнения этого задания будет продуман и реализован класс `Vector2D`, представляющий объект двухмерного вектора.

Свойство/метод	Описание
<code>Vector2D()</code>	Конструктор по умолчанию
<code>Vector2D(float x, float y);</code>	Конструктор вектора с координатами x и y
<code>Vector2D& Add(const Vector2D& vec);</code>	Сложение векторов
<code>Vector2D& Subtract(const Vector2D& vec);</code>	Разность векторов
<code>Vector2D& Multiply(const Vector2D& vec);</code>	Произведение векторов
<code>Vector2D& Divide(const Vector2D& vec);</code>	Частное векторов
<code>friend Vector2D& operator+(Vector2D& v1, Vector2D& v2);</code>	Перегрузка оператора сложения
<code>friend Vector2D& operator-(Vector2D& v1, Vector2D& v2);</code>	Перегрузка оператора вычитания
<code>friend Vector2D& operator*(Vector2D& v1, Vector2D& v2);</code>	Перегрузка оператора умножения
<code>friend Vector2D& operator/(Vector2D& v1, Vector2D& v2);</code>	Перегрузка оператора деления
<code>Vector2D& operator+=(Vector2D& vec);</code>	Перегрузка операторов сложного присваивания для суммы, разности, произведения и частного.
<code>Vector2D& operator-=(Vector2D& vec);</code>	
<code>Vector2D& operator*=(Vector2D& vec);</code>	
<code>Vector2D& operator/=(Vector2D& vec);</code>	
<code>Vector2D& operator*(& i);</code>	Перегрузка оператора для умножения на число
<code>friend std::ostream& operator<<(std::ostream& stream, Vector2D& vec);</code>	Перегрузка оператора вывода

Возможный вариант реализации:

```
#include "Vector2D.h"
Vector2D::Vector2D() {
    x = 0;
    y = 0;
}
Vector2D::Vector2D(float x, float y) {
    this->x = x;
    this->y = y;
}
Vector2D& Vector2D::Add(const Vector2D& vec) {
    this->x += vec.x;
    this->y += vec.y;

    return *this;
}
Vector2D& Vector2D::Subtract(const Vector2D& vec) {
    this->x -= vec.x;
    this->y -= vec.y;

    return *this;
}
Vector2D& Vector2D::Multiply(const Vector2D& vec) {
    this->x *= vec.x;
    this->y *= vec.y;

    return *this;
}
Vector2D& Vector2D::Divide(const Vector2D& vec) {
    this->x /= vec.x;
    this->y /= vec.y;

    return *this;
}

Vector2D& operator+(Vector2D& v1, const Vector2D& v2) {
    return v1.Add(v2);
}
Vector2D& operator-(Vector2D& v1, const Vector2D& v2) {
    return v1.Subtract(v2);
}
Vector2D& operator*(Vector2D& v1, const Vector2D& v2)
{
    return v1.Multiply(v2);
}
Vector2D& operator/(Vector2D& v1, const Vector2D& v2) {
    return v1.Divide(v2);
}

Vector2D& Vector2D::operator+=(const Vector2D& vec) {
    return this->Add(vec);
}
Vector2D& Vector2D::operator-=(const Vector2D& vec) {
    return this->Subtract(vec);
}
Vector2D& Vector2D::operator*=(const Vector2D& vec) {
    return this->Multiply(vec);
}
Vector2D& Vector2D::operator/=(const Vector2D& vec) {
    return this->Divide(vec);
}

Vector2D& Vector2D::operator*(const int& i) {
    this->x *= i;
    this->y *= i;

    return *this;
}

std::ostream& operator<<(std::ostream& stream, const Vector2D& vec) {
    stream << "(" << vec.x << "," << vec.y << ")";
    return stream;
}
```