Vektorok C++-ban

Krucsay András

2025. 03. 30.

A programokról általánosságban

A megoldás során két programot hoztunk létre. Az egyik a `vector2.h` header fájl, ahol a fő dolgok vannak, a másik a `main.cpp` fájl, ahol meg az ellenőrzés történik. Ez utóbbi kevésbé érdekes.

A `vector2.h` programról részletesebben

A teljes program a dokumentum végén található. A `#pragma once` feladata, hogy csak egyszer legyen a fordításkor beillesztve, jövőbeli hibákat kiküszöbölve. Sablonosztályt (`template` és `class`) definiálunk, amely két dimenziós vektort (`T x, y;`) reprezentál és általános típusa miatt bármilyen numerikus típus használható benne. Alapvetően egy $\begin{pmatrix}0\\0\end{pmatrix}$ vektort hozunk létre és utána töltjük bele az x és y értékeket. Ezután létrehozzuk az alábbi műveleteket:

- összeadás (`operator+` és `operator+=`)
- kivonás (`operator-` és `operator-=`)
- skalárral szorzás jobbról (`operator*` és `operator*=`)
- skalárral szorzás balról (`operator*`, mint barátművelet)
- skalárral osztás (`operator/` és `operator/=`)
- skaláris szorzat (`dot`)
- hossz meghatározása (`length`)
- hossznégyzet meghatározása (`sqlength`)

- normalizált vektor (`normalized`)

A legutolsó rész, avagy a második barátművelet a kiírásért felel.

A `main.cpp` programról részletesebben

A headerben definiáltakat alkalmazni kell a programban. Először magát a headert kell meghívni, amit a `#include "Vector2.h"` segítségével tehetünk meg. A fenti lista programba való meghívása ezen lista szerint történik v és v' vektorokkal, valamint f skalárral:

- összeadás: v + v' és v += v'
- kivonás: v - v' és v -= v'
- skalárral szorzás jobbról: v * f és v *= f
- skalárral szorzás balról: f * v
- skalárral osztás: v / f és v /= f
- skalárszorzás: v.dot(v')
- hossznégyzet: v.sqlength()
- hossz: v.length()
- normálás: v.normalized()

A programba táplált alap értékek szerint

$$v = v1 = \begin{pmatrix} 3 \\ 4 \end{pmatrix},\ v' = v2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix},\ f = 2$$

vagyis könnyen megállapíthatóak az eredmények manuálisan is. A program működésének helyessége könnyen megállapítható így.

Manuálisan az értékek: v+v' = $\begin{pmatrix} 4 \\ 6 \end{pmatrix}$, v-v' = $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$, v * f = $\begin{pmatrix} 6 \\ 8 \end{pmatrix}$, v / f = $\begin{pmatrix} 1.5 \\ 2 \end{pmatrix}$, v $\cdot$ v' = 11, $|\mathrm{v}|^2 = 25$, $|\mathrm{v}| = 5$, $\mathrm{e_v} = \begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix}$.

A program kimenete pedig a következő:
Összeg: (4, 6), Különbség: (2, 2), Skalárszorzás (jobbról): (6, 8), Skalárszorzás (balról): (6, 8), Skalárosztás: (1.5, 2), Skaláris szorzat (dot product): 11, v1 hosszának négyzete: 25, v1 hossza: 5, v1 normalizálva: (0.6, 0.8), v1 += v2: (4, 6), v1 -= v2: (3, 4), v1 *= 2: (6, 8), v1 /= 2: (3, 4) és ezek mind megfelelnek az elvárásoknak.

A programok

A vector2.h program

```
#pragma once
#include <cmath>
#include <iostream>

template <typename T>
class Vector2 {
public:
    T x, y;

    Vector2() : x(0), y(0) {}
    Vector2(T x, T y) : x(x), y(y) {}

    Vector2 operator+(const Vector2& other) const {
        return Vector2(x + other.x, y + other.y);
    }

    Vector2& operator+=(const Vector2& other) {
        x += other.x;
        y += other.y;
        return *this;
    }

    Vector2 operator-(const Vector2& other) const {
        return Vector2(x - other.x, y - other.y);
    }

    Vector2& operator-=(const Vector2& other) {
        x -= other.x;
        y -= other.y;
        return *this;
    }

    Vector2 operator*(T scalar) const {
        return Vector2(x * scalar, y * scalar);
    }

    Vector2& operator*=(T scalar) {
```

```
    x *= scalar;
    y *= scalar;
    return *this;
}

Vector2 operator/(T scalar) const {
    return Vector2(x / scalar, y / scalar);
}

Vector2& operator/=(T scalar) {
    x /= scalar;
    y /= scalar;
    return *this;
}

T dot(const Vector2& other) const {
    return x * other.x + y * other.y;
}

T sqlength() const {
    return x * x + y * y;
}

T length() const {
    return std::sqrt(sqlength());
}

Vector2 normalized() const {
    T len = length();
    if (len == 0) return Vector2(0, 0);
    return *this / len;
}

void normalize() {
    T len = length();
    if (len != 0) {
        *this /= len;
    }
}

friend Vector2 operator*(T scalar, const Vector2& vec) {
```

```
        return Vector2(vec.x * scalar, vec.y * scalar);
    }

    friend std::ostream& operator<<(std::ostream& os, const Vector2& vec) {
        os << "(" << vec.x << ", " << vec.y << ")";
        return os;
    }
};
```

A main.cpp program

```
#include <iostream>
#include "Vector2.h"

int main() {
    Vector2<float> v1(3.0f, 4.0f);
    Vector2<float> v2(1.0f, 2.0f);

    Vector2<float> sum = v1 + v2;
    std::cout << "Összeg: " << sum << std::endl;

    Vector2<float> diff = v1 - v2;
    std::cout << "Különbség: " << diff << std::endl;

    Vector2<float> scaled1 = v1 * 2.0f;
    Vector2<float> scaled2 = 2.0f * v1;
    std::cout << "Skalárszorzás (jobbról): " << scaled1 << std::endl;
    std::cout << "Skalárszorzás (balról): " << scaled2 << std::endl;

    Vector2<float> divided = v1 / 2.0f;
    std::cout << "Skalárosztás: " << divided << std::endl;

    float dotProduct = v1.dot(v2);
    std::cout << "Skaláris szorzat (dot product): " << dotProduct << std::endl;

    float sqlength = v1.sqlength();
    std::cout << "v1 hosszának négyzete: " << sqlength << std::endl;

    float length = v1.length();
    std::cout << "v1 hossza: " << length << std::endl;
```

```
    Vector2<float> normalized = v1.normalized();
    std::cout << "v1 normalizálva: " << normalized << std::endl;

    v1 += v2;
    std::cout << "v1 += v2: " << v1 << std::endl;

    v1 -= v2;
    std::cout << "v1 -= v2: " << v1 << std::endl;

    v1 *= 2.0f;
    std::cout << "v1 *= 2: " << v1 << std::endl;

    v1 /= 2.0f;
    std::cout << "v1 /= 2: " << v1 << std::endl;

    return 0;
}
```