```cpp
//////////////////////////////////////////////////////
// vec2.hpp
//////////////////////////////////////////////////////

#pragma once

#ifndef VEC2_HPP
#define VEC2_HPP

#include <ostream>

/** vec2 class model a 2D vector
    Vector function are provided as member functions. */
class vec2
{
    /** Internal x data*/
    float x_data;
    /** Internal y data*/
    float y_data;

public:
    /** Constructor to (0,0) vector*/
    vec2();
    /** Constructor to (x,y) vector*/
    vec2(float x_param,float y_param);

    /** get x value */
    float x() const;
    /** get y value */
    float y() const;
    /** set x value */
    float& x();
    /** set t value */
    float& y();

    /** get i_th value */
    float operator[](int k) const;
    /** set i_th value */
    float& operator[](int k);

    /** get i_th value */
    float operator()(int k) const;
    /** set i_th value */
    float& operator()(int k);


    /** Add v to current vector*/
    vec2& operator+=(vec2 const& v);
    /** Add two vectors*/
    vec2 operator+(vec2 const& v) const;

    /** Unary negation */
    vec2 operator-() const;

    /** Substract v to current vector*/
    vec2& operator-=(vec2 const& v);
    /** Substract two vectors*/
    vec2 operator-(vec2 const& v) const;

    /** Multiply scalar s to the current vector */
    vec2& operator*=(float s);
    /** Return multiplication of scalar s to the vector */
    vec2 operator*(float s) const;
```

```
    /** Divide current vector by the scalar s */
    vec2& operator/=(float s);
    /** Return vector divided by the scalar s */
    vec2 operator/(float s) const;

    /** Check if vector v equals the current vector */
    bool operator==(vec2 const& v) const;
    /** Check if vector v is not equal to the current vector */
    bool operator!=(vec2 const& v) const;

    /** Dot product between the current vector and v */
    float dot(vec2 const& v) const;
};

/** Return multiplication of scalar s to the vector
 * \note This cannot be done as a member function */
vec2 operator*(float s,vec2 const& v);

/** Export vector values to stream (to print for instance)
 * \note THis cannot be done as a member function */
std::ostream& operator<<(std::ostream& s,vec2 const& v);

#endif

////////////////////////////////////////////////////
// vec2.cpp
////////////////////////////////////////////////////

#include "vec2.hpp"

#include <cmath>
#include <iostream>

//constructor
vec2::vec2()
    :x_data(0.0f),y_data(0.0f) //use initilizer list
{}

//constructor
vec2::vec2(float x_param,float y_param)
    :x_data(x_param),y_data(y_param) //use initilizer list
{}

//get x
float vec2::x() const {return x_data;}
//get y
float vec2::y() const {return y_data;}
//set x
float& vec2::x() {return x_data;}
//set y
float& vec2::y() {return y_data;}

// get i_th value
float vec2::operator[](int k) const
{
    switch(k)
    {
    case 0:
        return x_data;
    case 1:
        return y_data;
    default:
        std::cerr<<"Erreur ["<<k<<"]"<<std::endl;
```

```
        exit(1);
    }
}

// set i_th value
float& vec2::operator[](int k)
{
    switch(k)
    {
    case 0:
        return x_data;
    case 1:
        return y_data;
    default:
        std::cerr<<"Erreur ["<<k<<"]"<<std::endl;
        exit(1);
    }
}

// get i_th value
float vec2::operator()(int k) const
{
    return (*this)[k]; //use already implemented function
}
// set i_th value
float& vec2::operator()(int k)
{
    return (*this)[k]; //use already implemented function
}

vec2& vec2::operator+=(vec2 const& v)
{
    x_data+=v.x();
    y_data+=v.y();

    return *this;
}

vec2 vec2::operator+(vec2 const& v) const
{
    vec2 temp=*this;
    temp+=v;
    return temp;
}

vec2& vec2::operator-=(vec2 const& v)
{
    x_data-=v.x();
    y_data-=v.y();

    return *this;
}
vec2 vec2::operator-(vec2 const& v) const
{
    vec2 temp=*this;
    temp-=v;
    return temp;
}

vec2& vec2::operator*=(float s)
{
    x_data*=s;
    y_data*=s;
    return *this;
```

```
}

vec2 vec2::operator*(float s) const
{
    vec2 temp=*this;
    temp*=s;
    return temp;
}

vec2& vec2::operator/=(float s)
{
    float epsilon=1e-6f;
    if(std::abs(s)<epsilon)
    {
        std::cout<<"Division par zero"<<std::endl;
        return *this;
    }

    x_data/=s;
    y_data/=s;

    return *this;
}

vec2 vec2::operator/(float s) const
{
    vec2 temp=*this;
    temp/=s;
    return temp;
}

vec2 vec2::operator-() const {return vec2(-x_data,-y_data);}

bool vec2::operator==(vec2 const& v) const
{
    float epsilon=1e-6f;
    if(std::abs(x_data-v.x())<epsilon &&
            std::abs(y_data-v.y())<epsilon)
        return true;
    return false;
}
bool vec2::operator!=(vec2 const& v) const
{
    return !(*this==v); //use already existing function
}

//dot product
float vec2::dot(vec2 const& v) const
{
    return x_data*v.x()+y_data*v.y();
}

//function
vec2 operator*(float s,vec2 const& v) {return v*s;}

//function
std::ostream& operator<<(std::ostream& s,vec2 const& v)
{
    s<<v.x()<<","<<v.y();
    return s;
}

////////////////////////////////////////////////////
// main.cpp
```

```cpp
//////////////////////////////////////////////////

#include "vec2.hpp"
#include <iostream>

int main()
{
    vec2 v0(6.4f,4.5f);
    vec2 v1(4.1f,2.5f);

    vec2 v2=v0+v1-vec2(v1.x(),v0[1]);

    std::cout<<"v2 = ("<<v2<<")"<<std::endl;

    float d=v2.dot(-v0);
    std::cout<<"v2.v0="<<d<<""<<std::endl;

    return 0;
}
```