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

#pragma once

#ifndef VEC2_HPP
#define VEC2_HPP

#include <ostream>

/** vec2 class model a 2D vector
    Vector function are provided as standard 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 v2 to v1*/
vec2& operator+=(vec2& v1,vec2 const& v2);
/** Add two vectors*/
vec2 operator+(vec2 const& v1,vec2 const& v2);

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

/** Substract v2 to v1*/
vec2& operator-=(vec2& v1,vec2 const& v2);
/** Substract two vectors*/
vec2 operator-(vec2 const& v1,vec2 const& v2);

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

```
/** Return multiplication of scalar s to the vector */
vec2 operator*(float s,vec2 const& v);

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

/** Check if two vectors are equal */
bool operator==(vec2 const& v1,vec2 const& v2);
/** Check if two vectors are not equal */
bool operator!=(vec2 const& v1,vec2 const& v2);

/** Export vector values to stream (to print for instance) */
std::ostream& operator<<(std::ostream& s,vec2 const& v);

/** Dot product between two vectors */
float dot(vec2 const& v1,vec2 const& v2);

#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& operator+=(vec2& v1,vec2 const& v2)
{
    v1.x()+=v2.x();
    v1.y()+=v2.y();

    return v1;
}
vec2 operator+(vec2 const& v1,vec2 const& v2)
{
    vec2 temp=v1;
    temp+=v2;
    return temp;
}
vec2& operator-=(vec2& v1,vec2 const& v2)
{
    v1.x()-=v2.x();
    v1.y()-=v2.y();

    return v1;
}
vec2 operator-(vec2 const& v1,vec2 const& v2)
{
    vec2 temp=v1;
    temp-=v2;
    return temp;
}

vec2& operator*=(vec2& v,float s)
{
    v.x()*=s;
    v.y()*=s;
    return v;
}

vec2 operator*(vec2 const& v,float s)
{
    vec2 temp=v;
    temp*=s;
```

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

    v.x()/=s;
    v.y()/=s;

    return v;
}
vec2 operator/(vec2 const& v,float s)
{
    vec2 temp=v;
    temp/=s;
    return temp;
}
vec2 operator*(float s,vec2 const& v) {return v*s;}
vec2 operator-(vec2 const& v) {return vec2(-v.x(),-v.y());}
bool operator==(vec2 const& v1,vec2 const& v2)
{
    float epsilon=1e-6f;
    if(std::abs(v1.x()-v2.x())<epsilon &&
            std::abs(v1.y()-v2.y())<epsilon)
        return true;
    return false;
}
bool operator!=(vec2 const& v1,vec2 const& v2)
{
    return !(v1==v2); //use already existing function
}
std::ostream& operator<<(std::ostream& s,vec2 const& v)
{
    s<<v.x()<<","<<v.y();
    return s;
}
//dot product
float dot(vec2 const& v1,vec2 const& v2)
{
    return v1.x()*v2.x()+v1.y()*v2.y();
}


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

    return 0;
}
```