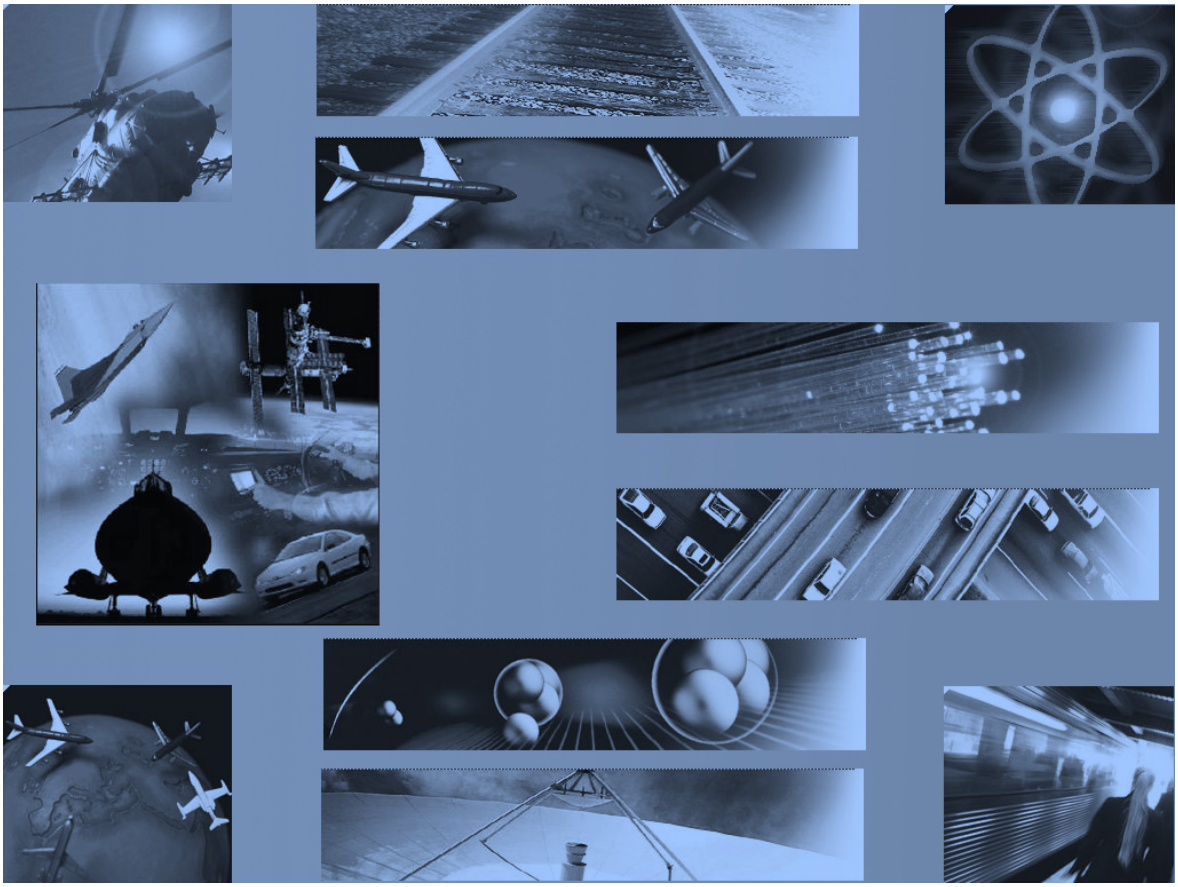
EJERCICIOS SEMANALES DE TIEMPO REAL



GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

C makes it easy to shoot yourself in the foot;

C++ makes it harder, but when you do, it blows your whole leg off

Bjarne Stroustrup (circa 1986)

EJERCICIOS DE LÓGICA DIFUSA (FUZZY LOGIC)

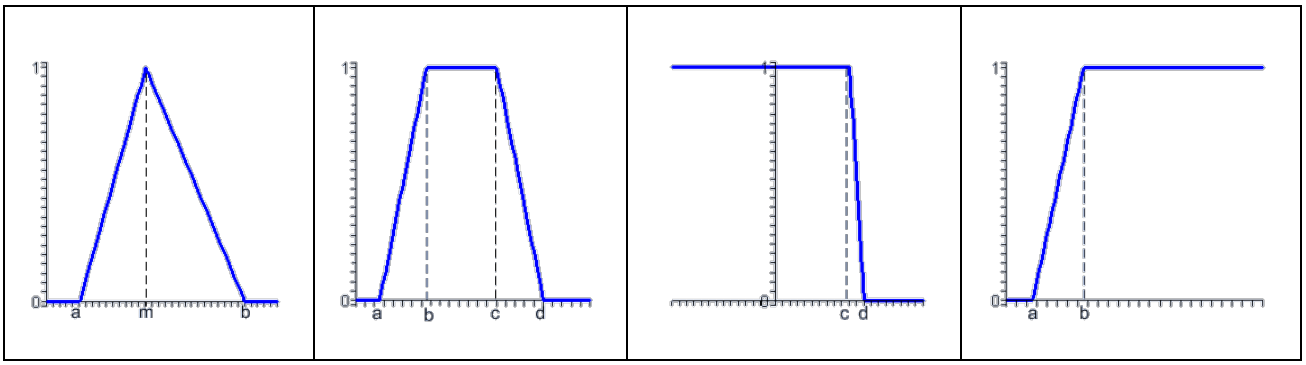
FUNCIONES DE PERTENENCIA (MEMBERSHIP FUNCTIONS)

La teoría de Lógica Difusa tiene aplicación en múltiples campos y es muy genérica. De entre todas las posibles aplicaciones, nosotros nos vamos a centrar en el diseño de controladores. Vea los siguientes videos para una introducción a *Fuzzy Logic* (Lógica Difusa, Lotfi Zadeh 1965).

- https://www.youtube.com/watch?v=J_Q5X0nTmrA
- https://www.youtube.com/watch?v=rln_kZbYaWc
- http://www.dma.fi.upm.es/recursos/aplicaciones/logica_borrosa/web/tutorial_fuzzy/
- La entrada de la Wikipedia en español
- http://www.maplesoft.com/applications/view.aspx?SID=1409&view=html
- http://petro.tanrei.ca/fuzzylogic/fuzzy_negnevistky.html

Nosotros vamos a utilizar los sistemas de **Sugeno**, ya que son más sencillos de implementar puesto que la última etapa, *defuzzyfication*, está basada en ecuaciones y no en conjuntos. Es ampliamente utilizado por su simplicidad computacional, sobre todo a la hora de diseñar controladores. El modelo general es el llamado modelo de Mamdami, en el que las salidas del sistema son también conjuntos difusos que deben "defuzificarse" para obtener un valor real (lo que se denomina un valor "*crisp*" en la terminología).

Vamos comenzar con la parte más sencilla: definición de las clases que modelan los conjuntos difusos. Los modelos más sencillos son el triángulo (definido por tres puntos), el trapecio, la 'S' hacia la izquierda y la 'S' hacia la derecha. En la figura se muestran gráficamente los cuatro conjuntos básicos:



Triángulo está definido por 3 puntos, Trapecio está definido por 4 puntos, 'S' hacia la derecha y hacia la izquierda están definidos por dos valores: inicio del valor '0 y del valor '1'. El siguiente código declara la clase Triangulo. Se pide que implemente dicha clase junto a las clases Trapecio, S_derecha y S_izquierda y pruebe que efectivamente funcionan correctamente.

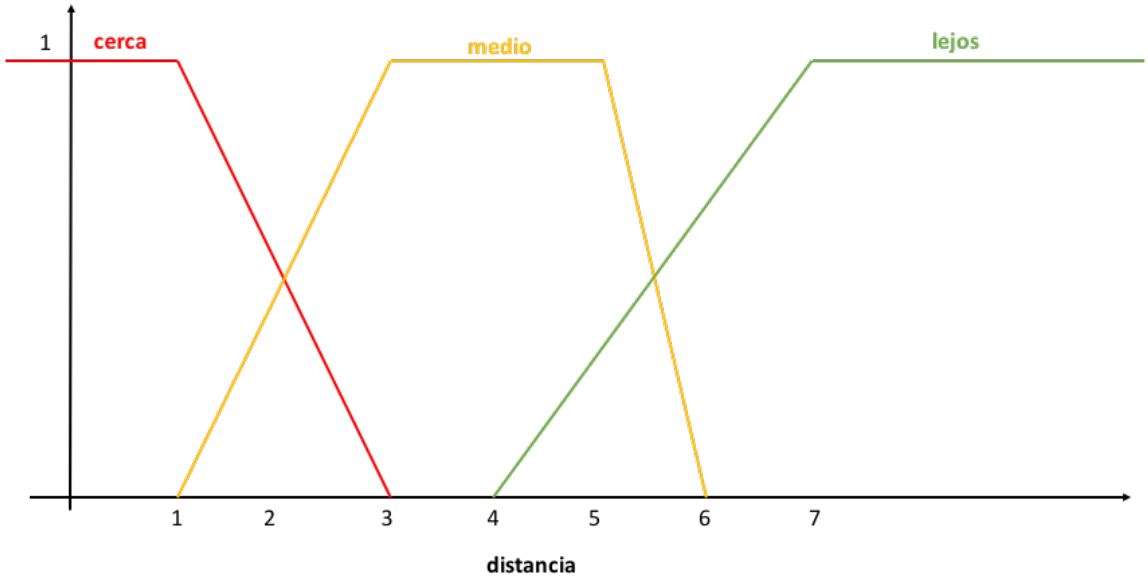
```
class Triangulo {
        double inicio_, vertice_, fin_;
public:
        friend std::ostream& operator<< (std::ostream& os, const Triangulo &t);
        Triangulo (double ini=-1, double vertice=0, double fin=1);
        // Retorna el valor difuso correspondiente a 'val'
        double fuzzyValue (double val);
};
```

CONTROLADOR DIFUSO 1

Se pide implementar el sistema de frenado de un coche con un controlador difuso. Para ello, se diseña el siguiente sistema lingüístico:

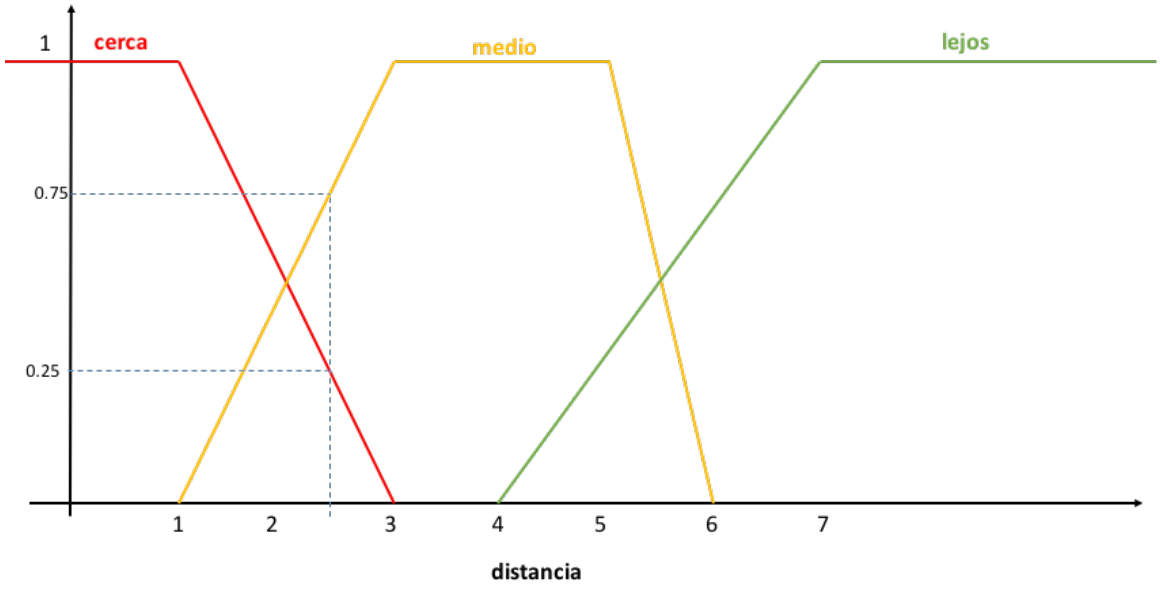
- Si el coche está cerca del siguiente, se acciona mucho el freno
- Si el coche está a distancia media del siguiente, se acciona el freno
- Si el coche está lejos del siguiente, se acciona poco el freno

La siguiente imagen muestra el sistema difuso correspondiente a "*distancia al coche siguiente*":



La acción "*poca*" del freno supone pulsarlo al 10%, la "*media*" al 40%, y la "*mucha*" al 90%.

Una vez definidos los conjuntos difusos y los valores de salida del controlador, vamos a verlo en funcionamiento. Supongamos que la distancia al coche siguiente medida en un instante dado fuera de 2.5 metros. Con este valor se obtienen los grados de pertenencia a los tres conjuntos difusos como muestra la siguiente figura:



Observe que el grado de pertenencia al conjunto "lejos" es creo en este caso aunque no aparece en la figura. Con los valores de pertenencia 0.25 a "cerca" y 0.75 a "medio", la acción de control se calcula de la siguiente manera:

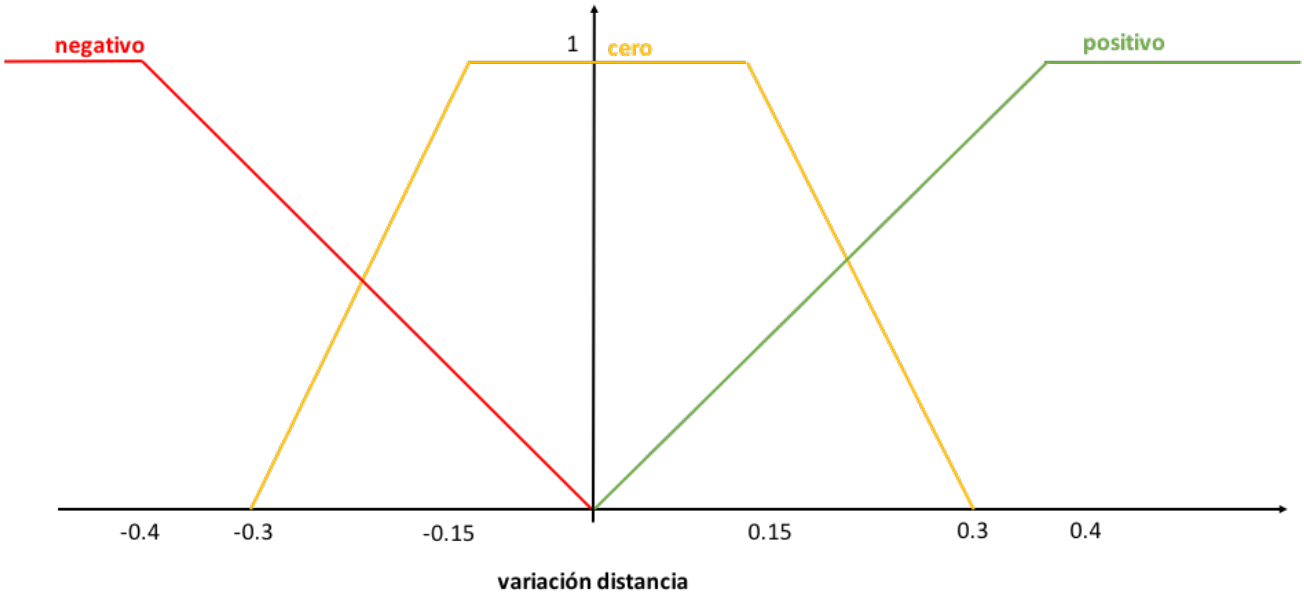
$$\frac{0.25 \cdot 90 + 0.75 \cdot 40}{0.25 + 0.75} = 52.5\%$$

Escriba un programa que implemente este controlador difuso y pruébelo con valores simulados introducidos por consola. Haga un bucle infinito que simule el controlador.

CONTROLADOR DIFUSO 2

En general, en los controladores difusos se tiene en cuenta no solo el error de la medida (en este ejemplo, la distancia entre dos coches) sino también la **variación del error** (derivada del error) puesto que da una idea de si, en este caso, el coche se está acercando (variación negativa) o alejando (variación positiva) del coche que tiene delante. De esta forma se puede diseñar un mejor controlador.

La siguiente imagen muestra el sistema difuso correspondiente a "variación de la distancia al coche siguiente":



Ahora entonces hay que definir un nuevo conjunto de reglas para tener en cuenta las posibles combinaciones entre las variables "distancia al coche siguiente" y "variación de la distancia al coche siguiente". La forma más habitual de mostrar las reglas en un controlador difuso de Sugeno es mediante una tabla como la siguiente:

		Variación distancia		
		Negativa	**Cero**	**Positiva**
Distancia	**Cerca**	90	70	50
	Medio	60	40	30
	Lejos	40	20	10

Por ejemplo, si la distancia es "Cerca" y la variación de la distancia es "Positiva", la acción de control será del 50%. Los valores de la acción de control por supuesto tienen que ser ajustados para el caso concreto. Sin embargo, como hemos visto, dado un valor de distancia y una variación, es posible que se active más de una regla, tal y como pasó en el ejemplo anterior. Para calcular ahora la acción de control final hay, según la teoría de lógica difusa, varias formas de proceder. De todas las posibles, emplearemos la siguiente:

Por cada regla activada, seleccionaremos el valor de pertenencia más pequeña de los dos conjuntos involucrados (distancia y variación de distancia). Este valor será el que se utilice para calcular la acción de control según la fórmula mostrada en el ejercicio anterior.

Por ejemplo, si la distancia es 2,5 metros, se tiene un grado de pertenencia 0.25 a "cerca" y 0.75 a "medio". Supongamos que la variación del error es en ese momento 0,23, lo que arroja un grado de pertenencia 0.46 a "cero" y 0.57 a "positiva". Con estos valores se activan las siguientes cuatro reglas:

1. Distancia "cerca" y variación "cero" (min(0.25, 0.46)=0.25)→ acción de control 70%
2. Distancia "cerca" y variación "positiva" (pertenencia 0.25)→ acción de control 50%

3. Distancia “medio” y variación “cero” (pertenencia 0.46)→ acción de control 40%
4. Distancia “medio” y variación “positiva” (pertenencia 0.57) → acción de control 30%

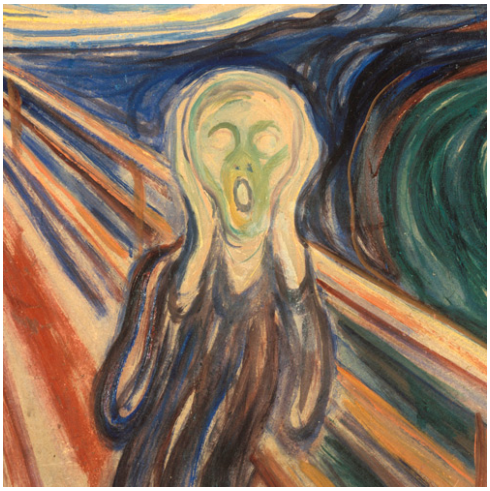
$$\frac{0.25 \cdot 70 + 0.25 \cdot 50 + 0.46 \cdot 40 + 0.57 \cdot 30}{0.25 + 0.25 + 0.46 + 0.57} = 42{,}81\%$$

Se pide que amplíe el programa anterior para implementar este controlador, que haga un pequeño “simulador” que solicite los datos por pantalla, y que implemente dos formas de combinar los valores de pertenencia de una regla activa: mínimo de los valores de pertenencia (como se propone) y producto de los valores de pertenencia. No hay un modelo mejor que otro. Intente que el código sea lo más reutilizable posible, por ejemplo, permitiendo que el programador elija la forma de combinar los valores.

Obviamente estos ejercicios son solo un ejemplo de un tipo de aplicación muy concreta que se puede desarrollar con lógica difusa, un controlador difuso. Dentro de los posibles tipos además se ha utilizado el más sencillo (Sugeno) para un ejemplo con una sola variable de control (distancia al coche). La lógica difusa es mucho más potente que esto. Pero espero que sea suficiente como para despertar su curiosidad.

Las soluciones vienen a continuación.

Consultar solo si ya está un poco desesperad@, o para comparar soluciones



CONJUNTOS BASICOS.HPP

```
#ifndef conjuntosBasicos_hpp
#define conjuntosBasicos_hpp

#include <iostream>

namespace upct { namespace tr { namespace fuzzy { namespace fuzzy_sets {



    class Triangulo {
        double inicio_, vertice_, fin_;
    public:
        friend std::ostream& operator<< (std::ostream& os, const Triangulo &t);
        Triangulo (double ini=-1, double vertice=0, double fin=1);
        double fuzzyValue (double val);
        ~Triangulo() = default;
    };

    class Trapecio {
        double inicio_, vertice1_, vertice2_, fin_;
    public:
        friend std::ostream& operator<< (std::ostream& os, const Trapecio &t);
        Trapecio (double ini=-2, double vertice1=-1, double vertice2=1, double fin=2);
        double fuzzyValue (double val);
    };

    class S_Izda {
        double vertice_, fin_;
    public:
        friend std::ostream& operator<< (std::ostream& os, const S_Izda &t);
        S_Izda (double vertice=-1, double fin=2);
        double fuzzyValue (double val);
    };

    class S_Dcha {
        double vertice_, ini_;
    public:
        friend std::ostream& operator<< (std::ostream& os, const S_Dcha &t);
        S_Dcha (double ini_=2, double vertice=-1);
        double fuzzyValue (double val);
    };



} } } }


#endif /* conjuntosBasicos_hpp */
```

CONJUNTOS BASICOS.CPP

```
#include "conjuntosBasicos.hpp"

namespace upct { namespace tr { namespace fuzzy { namespace fuzzy_sets {
```

```
    Triangulo::Triangulo (double ini, double vertice, double fin) {
            inicio_ = ini;
            vertice_ = vertice;
            fin_ = fin;
    }


    std::ostream& operator<< (std::ostream& os, const Triangulo &t) {
        os << "Fuzzy triangulo: ("<<t.inicio_<<") // ("<<t.vertice_<<") \\\\
("<<t.fin_<<')';
        return os;
    }

    double Triangulo::fuzzyValue (double val) {
        if (val<inicio_ or val>fin_)
            return 0.0;
        else if (val==vertice_)
            return 1;
        double pto0 = (val<vertice_? inicio_ : fin_);
        const double pediente = 1/(vertice_-pto0);
        const double ordenada = -pediente*pto0;
        return pediente*val+ordenada;

    }


    ////////////////////////////


    std::ostream& operator<< (std::ostream& os, const Trapecio &t) {
        os << "Fuzzy trapecio: ("<<t.inicio_<<") // (" << t.vertice1_ << ") == (" <<
t.vertice2_ << ") \\\\ (" << t.fin_ << ')';
        return os;
    }

    Trapecio::Trapecio (double ini, double vertice1, double vertice2, double fin) {
            inicio_ = ini;
            vertice1_ = vertice1;
            vertice2_ = vertice2;
    }

    double Trapecio::fuzzyValue (double val) {
        if (val<inicio_ or val>fin_)
            return 0.0;
        else if (val>=vertice1_ and val<=vertice2_)
            return 1;
        double pto0=inicio_, pto1=vertice1_;
        if (val>vertice2_) {
            pto0=fin_;
            pto1=vertice2_;
        }
        const double pediente = 1/(pto1-pto0);
        const double ordenada = -pediente*pto0;
        return pediente*val+ordenada;
    }

    ////////////////////////////


    S_Izda::S_Izda (double vertice, double fin) {
            vertice_ = vertice;
            fin_ = fin;
    }


    std::ostream& operator<< (std::ostream& os, const S_Izda &t) {
```

```
        os << "Fuzzy S_Izda: -- ("<<t.vertice_<<") \\\\ ("<<t.fin_<<')';
        return os;
    }

    double S_Izda::fuzzyValue (double val) {
        if (val<vertice_)
            return 1;
        if (val>fin_)
            return 0.0;
        const double pediente = 1/(vertice_-fin_);
        const double ordenada = -pediente*fin_;
        return pediente*val+ordenada;
    }

    ////////////////////////////


    S_Dcha::S_Dcha (double ini, double vertice) {
            vertice_ = vertice;
            ini_ = ini;
    }


    std::ostream& operator<< (std::ostream& os, const S_Dcha &t) {
        os << "Fuzzy S_Dcha:  // ("<<t.ini_<<") -- ("<<t.vertice_<<')';
        return os;
    }

    double S_Dcha::fuzzyValue (double val) {
        if (val>vertice_)
            return 1;
        if (val<ini_)
            return 0;
        const double pediente = 1/(vertice_-ini_);
        const double ordenada = -pediente*ini_;
        return pediente*val+ordenada;
    }


} } } }
```

MAIN

```
#include <iostream>
#include <limits>
#include <cmath>

#include "conjuntosBasicos.hpp"
namespace fuzzy_sets = upct::tr::fuzzy::fuzzy_sets;

void test_triangulo () {
    fuzzy_sets::Triangulo t {-5, 0, 5};
    std::cout << "FUZZY VALUE de "<<t<<std::endl;
    std::cout << "fValue(-10): " << t.fuzzyValue(-10) << ", fValue(10):
"<<t.fuzzyValue(10) << std::endl;
    std::cout << "fValue(-4): " << t.fuzzyValue(-4) <<", fValue(4): "<<t.fuzzyValue(4) <<
std::endl;
    std::cout << "fValue(-1): " << t.fuzzyValue(-1) <<", fValue(1): "<<t.fuzzyValue(1) <<
std::endl;
}
```

```
void test_trapecio() {
    fuzzy_sets::Trapecio t {-8,-3, 3, 8};
    std::cout << "FUZZY VALUE de "<<t<<std::endl;
    std::cout << "fValue(-10): " << t.fuzzyValue(-10) << ", fValue(10):
"<<t.fuzzyValue(10) << std::endl;
    std::cout << "fValue(-4): " << t.fuzzyValue(-4) <<", fValue(4): "<<t.fuzzyValue(4) <<
std::endl;
    std::cout << "fValue(-7): " << t.fuzzyValue(-7) <<", fValue(7): "<<t.fuzzyValue(7) <<
std::endl;
    std::cout << "fValue(-1): " << t.fuzzyValue(-1) <<", fValue(1): "<<t.fuzzyValue(1) <<
std::endl;
}

template <typename T>
void leer_descartando_entrada_incorrecta_mensaje
(const std::string &msg, T &var) {
    while (true) {
        std::cout << msg;
        std::cin >> var;
        if (std::cin.good()) break;
        std::cin.clear();
        std::cin.ignore(999, '\n');
    }
}


template <typename T>
void leer_descartando_entrada_incorrecta
(std::istream &is, T &var) {
    while (true) {
        is >> var;
        if (is.good() || is.eof()) break;
        is.clear();
        is.ignore(999, '\n');
    }
}



// cerca -> freno_mucho, medio -> freno_normal, lejos -> freno_poco
void ejemplo_freno_coche() {
    fuzzy_sets::Trapecio fuzzy_lejos{4.5, 6, 9, 120}, fuzzy_cerca {-1, 0, 1.5, 2.5};
    fuzzy_sets::Triangulo fuzzy_medio {2, 3.5, 5};
    double cerca, medio, lejos, distancia, freno;
    const double freno_poco=0.1, freno_normal=0.3, freno_mucho=0.8;
    for (size_t i=0; i<100; ++i) {
        leer_descartando_entrada_incorrecta_mensaje("Distancia al coche? ", distancia);
        cerca = fuzzy_cerca.fuzzyValue(distancia);
        medio = fuzzy_medio.fuzzyValue(distancia);
        lejos = fuzzy_lejos.fuzzyValue(distancia);
        freno =
(cerca*freno_mucho+medio*freno_normal+lejos*freno_poco)/(cerca+medio+lejos);
        std::cout << "Pulso " << freno*100 << "% del pedal de freno. Res: " << cerca <<
", " << medio << ", " << lejos << std::endl;
    }
}

void ej_coche () {
    fuzzy_sets::S_Izda cerca_mf {1,3};
    fuzzy_sets::Trapecio medio_mf {1,3,5,6};
    fuzzy_sets::S_Dcha lejos_mf {4,7};
    double pto = 2.5;
    double cerca = cerca_mf.fuzzyValue(pto);
    double medio = medio_mf.fuzzyValue(pto);
    std::cout << "Cerca: " << cerca <<" , medio: " << medio << std::endl;
```

```
    fuzzy_sets::S_Izda negativo_mf {-0.4, 0};
    fuzzy_sets::Trapecio cero_mf {-0.3, -0.15, 0.15, 0.3};
    fuzzy_sets::S_Dcha positivo_mf {0, 0.4};
    double error = 0.23;
    double cero = cero_mf.fuzzyValue(error);
    double posit = positivo_mf.fuzzyValue(error);
    std::cout << "cero: " << cero << ", posit: " << posit << std::endl;
}

int main(int argc, const char * argv[]) {
    //test_triangulo ();
    //test_trapecio();
    //ejemplo_freno_coche();
    ej_coche();
    std::cout << "Fin\n";
    return 0;
}
```