```
/**
* Prints Hello World.
**/
class Program
{
  public static void Main()
  {
    System.Console.WriteLine("Hello World!");
  }
}

#ifndef TFICHEROKEEL_H
#define TFICHEROKEEL_H

#include "defs.h" //(* #include...
#include <fstream>
#include <stdio.h>
#include <iostream>
using std::cout;
using std::endl;

//#include <forward_list>
//using std::forward_list;
#include <list>
using std::list;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <map>
using std::map;
//*)


/**  class TFicheroKEEL
  *
  *  Esta clase es una interfaz para utilizar los
     ficheros que pone a nuestra
  *  disposición el proyecto @link
     http://sci2s.ugr.es/keel/index.php KEEL @endlink
  *
  *  Extrae la información de los metadatos del fichero,
     lee la colección de datos y
  *  crea un fichero con el formato que necesita mi
     aplicación para gestionarlo con
  *  eficiencia:
```

```
  *
  *  - Se codifican los distintos valores de la clase
     con los códigos 0, 1...
  *  - Se codifican el resto de valores mediante números
     enteros consecutivos sin dejar
  *    ninguno reduciendo las necesidades de RAM de los
     algoritmos utilizados.
  *
  *  También crea el fichero D comprimido optimizando
     los códigos usados. Se guardan
  *  también las codificaciones hechas.
  *
  *  Guarda también todos los datos descriptivos del
     fichero, que ayudan a la toma de
  *  decisiones del analista y a la elaboración de
     informes para las pruebas que se
  *  hagan sobre estos ficheros.
  *
  *  Se leen líneas de un máximo de 4096 caracteres, si
     el fichero tuviera líneas más
  *  largas no será correcta la lectura y se podrán
     obtener resultados inesperados.
  *
  *  @todo Mayor control sobre capacidad_linea_ y
     capacidad_separador_ para no usar
  *        linea_ y posicion_separador_ fuera de su
     alcance.
  */
class TInfoFicheroKEEL;
class TFicheroKEEL
{
   public:
      static bool CompruebaSiEsKEEL(const string
         &nombre_fichero_datos)
      {
         FILE *fichero =
            fopen(nombre_fichero_datos.c_str(), "rt");
         if (!fichero)
         {
            cout << "No se ha podido abrir el fichero "
               << nombre_fichero_datos
                 << " (Abortada la lectura de fichero
                    KEEL)";
            fclose(fichero);
            return false;
         }
```

```
        // Busco @data, leyendo sólo las 500 primeras
           líneas
        int caracter = fgetc(fichero),
            num_linea = 0;
        while (caracter != EOF && num_linea < 500)
        {
           num_linea++;
           while (caracter != EOF && caracter != '\n'
              && caracter != '@')
              caracter = fgetc(fichero);
           if (caracter == '@')
           {
              caracter = fgetc(fichero);
              if (caracter == 'd') caracter =
                 fgetc(fichero); else continue;
              if (caracter == 'a') caracter =
                 fgetc(fichero); else continue;
              if (caracter == 't') caracter =
                 fgetc(fichero); else continue;
              if (caracter == 'a') caracter =
                 fgetc(fichero); else continue;
              // Si lee @data termina el bucle y la
                 búsqueda
              break;
           }
           caracter = fgetc(fichero);
        }
        fclose(fichero);
        return (caracter != EOF && num_linea < 500);
     }

  private:
     TFicheroKEEL(const string &ruta_ficheros_OUT,
        const string &nombre_fichero_KEEL);
     virtual ~TFicheroKEEL();

     bool LeeMetadatos(); //(* Métodos de lectura del
        fichero KEEL
     bool LeeEtiqueta();
     bool LeeNombre();
     bool LeeTipoYDominio();
     bool LeeMetadato();
     bool LeeAtributo();
     bool LeeInputOutput();
```

```
      bool LeeDatos();
      bool LeeRegistro(); //*)

//      unsigned long GetNumRegistros() { return
    num_registros_; }
      unsigned long GetNumVariables() { return
          nombre_variables_.size(); }
      const int GetNumClases() const { return
          num_clases_; }
//      unsigned long GetNumValores() { return
    num_valores_; }
      vector<string> *GetNombreVariables() { return
          &nombre_variables_; }

      unsigned long GuardaD();
      unsigned long GuardaDComprimido(const string
          &nombre_fichero_D);
//      unsigned long GuardaC1();

      const bool Codificado() const { return
          codificado_; }

      void MuestraElRestoDeLinea();
      int SaltaEspaciosYComas();

      void ReorganizaVariables(); //!< Coloca las clases
          en primer lugar

      unsigned long Codifica();
      int BuscaDatos();
      int LeeYGuardaRegistro(std::ofstream &fichero_OUT);

   private:
      //(* Miembros privados
      string carpeta_proyecto_; //!< Donde guardar
          ficheros auxiliares
      string nombre_fichero_KEEL_; //!< Nombre y
          ubicación del fichero
      FILE *fichero_; //!< Fichero KEEL

      int num_variables_,
          num_clases_;
      unsigned long num_registros_; //!< Número de
          registros del fichero
      unsigned long num_valores_; //!< Número de valores
          distintos en el fichero
```

```
        /// @todo Aclarar si uso list o vector en TODOS
            los miembros.
        /// @todo Sustituir por TAtributo
        vector<string> nombre_variables_; //!< Nombres de
            las variables
        vector< vector<string> > dominio_variables_; //!<
            Dominio teórico de variables
        vector<char> tipo_variables_; //!< Real, entero o
            categórico

        map<string, unsigned long> **valores_; //!<
            Valores leídos en el fichero

        vector<string> input_, //!< Nombre de los atributos
                       output_; //!< Nombres de las clases

        string nombre_coleccion_;

        char tipo_metadato_;

        string *codigo_2_valor_;
        map<string, int> **valor_2_codigo_;
        bool codificado_; //*)

    friend class TInfoFicheroKEEL;
};

#endif // TFICHEROKEEL_H
```