```hpp
/* ****************** LE HPP AVEC LA DECLARATION D'UNE CLASS ****************** */

#include <string> //pour les string
#include <iostream> //pour l'affichage cout cin ...

#ifndef MACLASS_HPP
# define MACLASS_HPP

class MaClass1; //declaration d-une class utilisee dans la class mais sans
                // affiliation

class   MaClass
{
    private : // accessible que par la classe elle-meme

        int         _unChiffre;
        std::string _uneString;

    protected : //accessible que par la classe et ses enfants

        // virtual = va chercher la fonction de la derniere generation
        virtual void    uneFonction() const;

    public : //accessible directement par tous


        //definition d'une exception
        class MonException : public std::exception
        {
            public :
                virtual const char *what() const throw();
        };

        MaClass(); //constructeur par defaut sans argument
        MaClass(int unNombre, std::string duTexte); //constructeur avec arguments
        MaClass(const MaClass &uneInstance); //constucteur par copie
        MaClass &operator=( const MaClass &rhs); //surcharge de l'op. =
        // rhs : right hand side : ce qui est situe a droite du =
        ~MaClass(); //destructeur;

        int         unChiffrePublique;

        //fonctions publiques = methodes publique
        std::string getString() const;
        //const a la fin annonce que la fonction ne modifie pas les variables
        // et empeche la compilation si une modif tente d'etre faite.
        void        setUnChiffre(int n);
        void        actionOne(MaClass1 const &class1);
};

std::ostream &operator<<(std::ostream &o, const MaClass &uneInstance);

#endif
```

```cpp
/* ********* UTILISATION ********* */

int main()
{
    MaClass MonInstance(82, "lol");

    MonInstance.getString();
    return 0;
}
```

```cpp
const char *MaClass::MonException::what() const throw()
{
    return ("mon super message d'erreur");
}
```

```cpp
// constructeur par defaut
MaClass::MaClass()
    : _unChiffre(42), _uneString("quelle ecole!"), unChiffrePublique(0)
{}

// plus ou moins equivalent a (moins optimum car fait en 2 temps):
MaClass::MaClass()
{
    this->_unChiffre = 42;
    this->_uneString = "quelle ecole!";
    this->unChiffrePublique = 0;
}

//constructeur avec argument avec une condition sur un des arguments
MaClass::MaClass(int unNombre, std::string duTexte)
    :_uneString(duTexte), unChiffrePublique(0)
{
    if (unNombre > 10)
        this->_unChiffre = unNombre;
    else
        this->_unChiffre = 42;
}

//constucteur par copie
MaClass::MaClass(const MaClass &uneInstance)
    :_unChiffre(uneInstance._unChiffre)
{
    *this = uneInstance; // fait appel a l'operateur = qu'on surcharge
}

//destructeur
MaClass::~MaClass()
{
    //ne pas oublier les "delete" s'il y a eu des new dans le constructeur!
}
```

```cpp
// surcharge de l'operateur =
MaClass &MaClass::operator=( const MaClass &rhs)
{

    if (this != &rhs)
    {
        this->_uneString = rhs._uneString;
        this->unChiffrePublique = rhs.unChiffrePublique;
    }
    return (*this);
}
```

```cpp
std::string MaClass::getString() const
{
    return (this->_uneString);
}

void    MaClass::setUnChiffre(int n)
{
    this->_unChiffre = n;
}
```

```cpp
// surcharge de l'operateur "<<" de ostream. permet de retourner un texte sur
// les sorties appelee
std::ostream &operator<<(std::ostream &o, const MaClass &uneInstance)
{
    o << uneInstance.getString() << " oui c'est clair " ;
    return (o);
}
```