```cpp
#include "Carre.hpp"

void Carre::setCote(const double c) {
  super::setLargeur(c);
  super::setLongueur(c);
}

double Carre::getCote() const {
  return super::getLargeur();
}

std::ostream&operator<<(std::ostream &f, const Carre& c) {
  return f << "carre(" << c.getCote()  << ")";
}

std::string Carre::quiSuisJe() const {
  return "un carré";
}
```



```cpp
#pragma once

#include <cassert>

#include <string>
#include "Rectangle.hpp"

class Carre : public Rectangle {
  typedef Rectangle super;
  // invariant de classe : largeur, longueur>=0 et largeur==longueur
public:

  int x =19;

  void setLargeur(const double l)  =delete;
  void setLongueur(const double L) =delete;
  double getLargeur()  const =delete;
  double getLongueur() const =delete;

  Carre(const double c=0) : Rectangle(c,c) { }
  Carre(const Rectangle &r) : Rectangle(r.getLargeur(), r.getLargeur()) {
    assert(r.getLargeur()== r.getLargeur());
  }

  //  le destructeur (pédagogique)
  ~Carre() {
#ifdef DEBUG
    std::cout << "destructeur Carre" << std::endl;
#endif
  }

  void setCote(const double c);
  double getCote() const;

  std::string quiSuisJe() const override;

  friend std::ostream&operator<<(std::ostream &f, const Carre& c);
};
```

```cpp
#include "Cercle.hpp"

void Cercle::setRayon(const double r) {
  super::setPetitRayon(r);
  super::setGrandRayon(r);
}

double Cercle::getRayon() const {
  return super::getPetitRayon();
}

std::ostream&operator<<(std::ostream &f, const Cercle& c) {
  return f << "cercle(" << c.getRayon()  << ")";
}

std::string Cercle::quiSuisJe() const {
  return "un cercle";
}
```



```cpp
#pragma once

#include <cassert>
#include <string>
#include "Ellipse.hpp"

class Cercle : public Ellipse {
  // invariant : petitRayon, grandRayon>=0 et petitRayon=grandRayon
  typedef Ellipse super;
private:
  double getPetitRayon() const;
  double getGrandRayon() const;
  void setPetitRayon(const double pa);
  void setGrandRayon(const double ga);

public:
  Cercle(const double r=0) : Ellipse(r,r) {
#ifdef DEBUG
    assert(r>=0);
#endif
  }

  Cercle(const Ellipse &e) : Ellipse(e.getPetitRayon(), e.getPetitRayon()) {
    assert(e.getPetitRayon() == e.getPetitRayon());
  }

  //  le destructeur (pédagogique)
  ~Cercle() {
#ifdef DEBUG
    std::cout << "destructeur Cercle" << std::endl;
#endif
  }

  void setRayon(const double r);
  double getRayon() const;
  std::string quiSuisJe() const override;

  friend std::ostream&operator<<(std::ostream &f, const Cercle& c);
};
```

```cpp
#include "Ellipse.hpp"
#include <cmath>


double Ellipse::getPetitRayon() const {
  return this->petitRayon;
}

double Ellipse::getGrandRayon() const {
  return this->grandRayon;
}

void Ellipse::setPetitRayon(const double pa) {
  this->petitRayon = pa;
}

void Ellipse::setGrandRayon(const double ga) {
  this->grandRayon = ga;
}

double Ellipse::perimetre() const  {
  return M_PI*sqrt(2*((this->petitRayon*this->petitRayon)+
                      (this->grandRayon*this->grandRayon)));
}

double Ellipse::surface() const {
  return M_PI * this->petitRayon * this->grandRayon;
}

std::ostream&operator<<(std::ostream &f, const Ellipse& e) {
  return f << "ellipse(" << e.petitRayon  << "," << e.grandRayon << ")";
}

std::string Ellipse::quiSuisJe() const {
  return "une ellipse";
}
```



```cpp
#pragma once

#ifdef DEBUG
#include <cassert>
#endif

#include <iostream>
#include <string>
#include <cmath>
#include "Figure.hpp"

class Ellipse : public Figure {
/* Invariant : petitRayon>=0, grandRayon>=0 */

protected:
  double petitRayon;
  double grandRayon;

public:
  Ellipse(const double pr=0, const double gr=0) :
    petitRayon(pr), grandRayon(gr) {
#ifdef DEBUG
    assert(pr>=0 && gr>=0);
#endif
  }

#ifdef DEBUG
  //  le destructeur (pédagogique)
  ~Ellipse() {
    std::cout << "destructeur Ellipse" << std::endl;
  }
#endif

  double getPetitRayon() const;
  double getGrandRayon() const;

  void setPetitRayon(const double pa);
  void setGrandRayon(const double ga);

  double perimetre() const override;
  double surface() const override;

  std::string quiSuisJe() const override;

  friend std::ostream&operator<<(std::ostream &f, const Ellipse& e);
};
```

```cpp
#include "Figure.hpp"

std::string Figure::quiSuisJe() const {
  return "une figure";
}
```



```cpp
#pragma once

#include <string>
#include <iostream>

class Figure {
public:
  //  le destructeur (pédagogique)
#ifdef DEBUG
  virtual  ~Figure() {
    std::cout << "destructeur Figure" << std::endl;
  }
#endif

  virtual std::string quiSuisJe() const;
  virtual double surface() const =0;
  virtual double perimetre() const =0;
};
```

```cpp
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <vector>

#include "Carre.hpp"
#include "Cercle.hpp"

void afficher(const Figure *f) {
  std::cout << "Je suis " << f->quiSuisJe();
}

int main() {
  /*
  std::vector<Figure *> vf = { new Cercle(3), new Rectangle(3,1) };
  for (auto f : vf) {
    afficher(f);
    std::cout << " de surface = " << f->surface() << std::endl;
  }

  Carre c(10);
  std::cout << "c = " << c.getCote() << std::endl;
  */
  Rectangle r1(12,15);
  Carre c1(10);
  //  Carre c2 = (Carre) r1;
  r1 = c1;
  std::cout << "r1 = " << r1 << std::endl;
  //std::cout << "c2.x = " << c2.x << std::endl;
  /*  Carre c3 = c1;

  r1.setLargeur(3);

  std::cout << "c2 = " << c2 << std::endl;
  std::cout << "c3 = " << c3 << std::endl;
  */
  return EXIT_SUCCESS;
}
```



```cpp
#include "Rectangle.hpp"


double Rectangle::getLargeur() const {
  return this->largeur;
}

double Rectangle::getLongueur() const {
  return this->longueur;
}

void Rectangle::setLargeur(const double l) {
  this->largeur = l;
}

void Rectangle::setLongueur(const double L) {
  this->longueur = L;
}

double Rectangle::perimetre() const {
  return 2*(this->largeur+this->longueur);
}

double Rectangle::surface() const  {
  return this->largeur*this->longueur;
}

std::ostream&operator<<(std::ostream &f, const Rectangle& r) {
  return f << "rectangle(" << r.largeur  << "," << r.longueur << ")";
}

std::string Rectangle::quiSuisJe() const {
  return "un rectangle";
}
```

```cpp
#pragma once

#ifdef DEBUG
#include <cassert>
#endif

#include <string>
#include <iostream>
#include "Figure.hpp"

class Rectangle : public Figure {
/* Invariant : largeur>=0, longueur>=0 */
protected:
  double largeur;
  double longueur;

public:

  Rectangle(const double l=0, const double L=0) :
    largeur(l), longueur(L)
  {
#ifdef DEBUG
    assert(l>=0 && L>=0);
#endif
  }

#ifdef DEBUG
  //  le destructeur (pédagogique)
  ~Rectangle() {
    std::cout << "destructeur Rectangle" << std::endl;
  }
#endif

  double getLargeur() const;
  double getLongueur() const;

  void setLargeur(const double l);
  void setLongueur(const double L);

  double perimetre() const override;
  double surface() const override;

  std::string quiSuisJe() const override;

  friend std::ostream&operator<<(std::ostream &f, const Rectangle& c);
};
```