TP 4 - Problem 4 - I

```cpp
#include <cmath>
#include <exception>
#include <iostream>
#include <numbers>
#include <vector>

#define SHOW(arg) std::cout << "Macro SHOW “" #arg "”: " << (arg) << '\n';

// A class to represent a point.
class Point {
public:
  // Default ctor: initialize the data members to Not A Number.
  Point() : x_{NAN}, y_{NAN} {}
  // Parametrised ctor.
  Point(double x, double y) : x_{x}, y_{y} {}
  // Compute the length between two points.
  double length(Point other) const {
    double const dx{x_ - other.x_};
    double const dy{y_ - other.y_};
```

TP 4 - Problem 4 - II

```
    return std::sqrt(dx * dx + dy * dy);
  }

private:
  double x_;
  double y_;
  // The free operator << must be a friend of my user defined class so it
  // have access to the private data member of this class. It is preferable
  // to declare the operator as a private one: it is only found via the
  // argument-dependant lookup.
  friend std::ostream &operator<<(std::ostream &, Point const &);
};

// The free function to get an external representation of a point.
std::ostream &operator<<(std::ostream &os, Point const &point) {
  os << '(' << point.x_ << ' ' << point.y_ << ')';
  return os;
}

// Abstract base class, i.e. the specification of a interface.
```

TP 4 - Problem 4 - III

```
struct Shape {
  virtual void print() const = 0;
  virtual double area() const = 0;
  virtual double perimeter() const = 0;
  virtual ~Shape(){};
};

class Triangle : public Shape {
public:
  // Default ctor.
  Triangle()
      : point_1_{}, point_2_{}, point_3_{}, side_1_{NAN}, side_2_{NAN},
        side_3_{NAN} {}
  // Parametrised ctor.
  Triangle(Point const &point_1, Point const &point_2,
           Point const &point_3)
      : point_1_{point_1}, point_2_{point_2}, point_3_{point_3},
        side_1_{point_1.length(point_2)}, side_2_{point_2.length(point_3)},
        side_3_{point_3.length(point_1)} {}
  // Dtor.
```

TP 4 - Problem 4 - IV

```
  ~Triangle() {}
  void print() const override {
    std::cout << "Triangle{" << point_1_ << ' ' << point_2_ << ' '
              << point_3_ << "}\n";
  }
  double area() const override {
    if (CheckIfTriangle()) {
      // Using Heron's formula.
      double const s{(side_1_ + side_2_ + side_3_) / 2};
      return std::sqrt(s * (s - side_1_) * (s - side_2_) * (s - side_3_));
    } else
      return NAN;
  }
  double perimeter() const override {
    return CheckIfTriangle() ? side_1_ + side_2_ + side_3_ : NAN;
  }
  void SetPoints(Point const &point_1, Point const &point_2,
                 Point const &point_3) {
    point_1_ = point_1;
    point_2_ = point_2;
```

TP 4 - Problem 4 - V

```
    point_3_ = point_3;
    side_1_ = point_1.length(point_2);
    side_2_ = point_2.length(point_3);
    side_3_ = point_3.length(point_1);
  }
  double side_1() const { return side_1_; }
  double side_2() const { return side_2_; }
  double side_3() const { return side_3_; }

private:
  bool CheckIfTriangle() const {
    if (std::isnan(side_1_) || std::isnan(side_2_) || std::isnan(side_3_))
      return false;
    return (side_1_ + side_2_ > side_3_) &&
           (side_1_ + side_3_ > side_2_) && (side_2_ + side_3_ > side_1_);
  }
  Point point_1_;
  Point point_2_;
  Point point_3_;
  double side_1_;
```

TP 4 - Problem 4 - VI

```cpp
  double side_2_;
  double side_3_;
};

class Rectangle : public Shape {
public:
  // Default ctor.
  Rectangle() : point_{}, width_{NAN}, height_{NAN} {}
  // Parametrised ctor.
  Rectangle(Point const &point, double width, double height)
      : point_{point}, width_{width}, height_{height} {
    if (width <= 0)
      throw std::domain_error{"Rectangle: the width must be positive."};
    if (height <= 0)
      throw std::domain_error{"Rectangle: the height must be positive."};
  }
  // Dtor.
  ~Rectangle() {}
  void print() const override {
    std::cout << "Rectangle{" << point_ << ' ' << width_ << ' ' << height_
```

TP 4 - Problem 4 - VII

```cpp
              << "}\n";
  }
  double area() const override { return width_ * height_; }
  double perimeter() const override { return 2 * (width_ + height_); }
  void SetWidth(double width) {
    if (width <= 0)
      throw std::domain_error{
          "Rectangle::SetWidth: the width must be positive."};
    width_ = width;
  }
  void SetHeight(double height) {
    if (height <= 0)
      throw std::domain_error{
          "Rectangle::SetHeight: the height must be positive."};
    height_ = height;
  }
  Point GetCorner() const { return point_; }
  double GetWidth() const { return width_; }
  double GetHeight() const { return height_; }

```

TP 4 - Problem 4 - VIII

```cpp
private:
  Point const point_;
  double width_;
  double height_;
};

class Circle : public Shape {
public:
  // Parametrised ctor.
  Circle(Point const &point, double radius)
      : point_{point}, radius_{radius} {
    if (radius <= 0)
      throw std::domain_error{"Circle: the radius must be positive."};
  }
  // Dtor.
  ~Circle() {}
  void print() const override {
    std::cout << "Circle{" << point_ << ' ' << radius_ << "}\n";
  }
  double area() const override {
```

TP 4 - Problem 4 - IX

```cpp
    return std::numbers::pi * radius_ * radius_;
  }
  double perimeter() const override {
    return 2 * std::numbers::pi * radius_;
  }

private:
  Point const point_;
  double radius_;
};

class GoodSquare : public Shape {
public:
  // Parametrised ctor.
  GoodSquare(Point const &point, double size) : rect_{point, size, size} {
    if (size <= 0)
      throw std::domain_error{"GoodSquare: the size must be positive."};
  }
  // Dtor.
  ~GoodSquare() {}
```

TP 4 - Problem 4 - X

```
  void print() const override {
    std::cout << "GoodSquare{" << rect_.GetCorner() << ' '
              << rect_.GetWidth() << "}\n";
  }
  double area() const override {
    return rect_.GetWidth() * rect_.GetWidth();
  }
  double perimeter() const override { return 4 * rect_.GetWidth(); }

private:
  Rectangle rect_;
};
void characteristics(Shape const *v[], int n) {
  std::cout << "\nC-style characteristics function output:\n";
  for (int i{}; i < n; ++i) {
    v[i]->print();
    std::cout << "area: " << v[i]->area()
              << " perimeter: " << v[i]->perimeter() << '\n';
  }
}
```

TP 4 - Problem 4 - XI

```cpp
void characteristics(std::vector<Shape const *> v) {
  std::cout << "\nSTL-style characteristics function output:\n";
  for (Shape const *shape_ptr : v) {
    shape_ptr->print();
    std::cout << "area: " << shape_ptr->area()
              << " perimeter: " << shape_ptr->perimeter() << '\n';
  }
}

int main() {
  SHOW(Point{}.length(Point{}))
  SHOW((Point{0, 0}.length(Point{0, 1})))
  SHOW((Point{0, 0}.length(Point{1, 0})))
  SHOW((Point{0, 0}.length(Point{1, 1})))
  Triangle{}.print();
  // Not a triangle.
  SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{0, 2}}.perimeter()))
  SHOW((Triangle{Point{0, 0}, Point{1, 1}, Point{2, 2}}.perimeter()))
  // Triangle.
  SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.perimeter()))
```

TP 4 - Problem 4 - XII

```
    SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.area()))
    // Rectangle
    Rectangle{}.print();
    Rectangle{Point{1, 1}, 2, 3}.print();
    SHOW((Rectangle{Point{1, 1}, 2, 3}.perimeter()))
    SHOW((Rectangle{Point{1, 1}, 2, 3}.area()))
    // Circle
    Circle{Point{0, 0}, 1}.print();
    SHOW((Circle{Point{0, 0}, 1}.area()));
    SHOW((Circle{Point{0, 0}, 1}.perimeter()));
    // GoodSquare
    GoodSquare{Point{0, 0}, 2}.print();
    SHOW((GoodSquare{Point{0, 0}, 2}.area()));
    SHOW((GoodSquare{Point{0, 0}, 2}.perimeter()));

    Triangle my_triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}};
    Rectangle my_rectangle{Point{1, 1}, 2, 3};
    Circle my_circle{Point{0, 0}, 1};
    GoodSquare my_square{Point{0, 0}, 2};
    {
```

TP 4 - Problem 4 - XIII

```
    // C style array.
    Shape const *my_menagerie[]{&my_triangle, &my_rectangle, &my_circle,
                                &my_square};
    int constexpr my_menagerie_sz{sizeof my_menagerie /
                                  sizeof my_menagerie[0]};
    characteristics(my_menagerie, my_menagerie_sz);
  }
  {
    // STL style array.
    std::vector<Shape const *> my_menagerie{&my_triangle, &my_rectangle,
                                            &my_circle, &my_square};
    characteristics(my_menagerie);
  }
  return 0;
}
```

TP 4 - Problem 4 - XIV

Output:

```
Macro SHOW “Point{}.length(Point{})”: nan
Macro SHOW “(Point{0, 0}.length(Point{0, 1}))”: 1
Macro SHOW “(Point{0, 0}.length(Point{1, 0}))”: 1
Macro SHOW “(Point{0, 0}.length(Point{1, 1}))”: 1.41421
Triangle{(nan nan) (nan nan) (nan nan)}
Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{0,
 ↪  2}}.perimeter())”: nan
Macro SHOW “(Triangle{Point{0, 0}, Point{1, 1}, Point{2,
 ↪  2}}.perimeter())”: nan
Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{1,
 ↪  0}}.perimeter())”: 3.41421
Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.area())”:
 ↪  0.5
Rectangle{(nan nan) nan nan}
Rectangle{(1 1) 2 3}
Macro SHOW “(Rectangle{Point{1, 1}, 2, 3}.perimeter())”: 10
Macro SHOW “(Rectangle{Point{1, 1}, 2, 3}.area())”: 6
Circle{(0 0) 1}
```

TP 4 - Problem 4 - XV

```
Macro SHOW “(Circle{Point{0, 0}, 1}.area())”: 3.14159
Macro SHOW “(Circle{Point{0, 0}, 1}.perimeter())”: 6.28319
GoodSquare{(0 0) 2}
Macro SHOW “(GoodSquare{Point{0, 0}, 2}.area())”: 4
Macro SHOW “(GoodSquare{Point{0, 0}, 2}.perimeter())”: 8

C-style characteristics function output:
Triangle{(0 0) (0 1) (1 0)}
area: 0.5 perimeter: 3.41421
Rectangle{(1 1) 2 3}
area: 6 perimeter: 10
Circle{(0 0) 1}
area: 3.14159 perimeter: 6.28319
GoodSquare{(0 0) 2}
area: 4 perimeter: 8

STL-style characteristics function output:
Triangle{(0 0) (0 1) (1 0)}
area: 0.5 perimeter: 3.41421
Rectangle{(1 1) 2 3}
```

TP 4 - Problem 4 - XVI

```
area: 6 perimeter: 10
Circle{(0 0) 1}
area: 3.14159 perimeter: 6.28319
GoodSquare{(0 0) 2}
area: 4 perimeter: 8
```