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

```
20        return std::sqrt(dx * dx + dy * dy);
21      }
22
23    private:
24      double x_;
25      double y_;
26      // The free operator << must be a friend of my user defined class so it
27      // have access to the private data member of this class. It is preferable
28      // to declare the operator as a private one: it is only found via the
29      // argument-dependant lookup.
30      friend std::ostream &operator<<(std::ostream &, Point const &);
31    };
32
33    // The free function to get an external representation of a point.
34    std::ostream &operator<<(std::ostream &os, Point const &point) {
35      os << '(' << point.x_ << ' ' << point.y_ << ')';
36      return os;
37    }
38
39    // Abstract base class, i.e. the specification of a interface.
```

TP 4 - Problem 4 - III

```
40  struct Shape {
41    virtual void print() const = 0;
42    virtual double area() const = 0;
43    virtual double perimeter() const = 0;
44    virtual ~Shape(){};
45  };
46
47  class Triangle : public Shape {
48  public:
49    // Default ctor.
50    Triangle()
51        : point_1_{}, point_2_{}, point_3_{}, side_1_{NAN}, side_2_{NAN},
52          side_3_{NAN} {}
53    // Parametrised ctor.
54    Triangle(Point const &point_1, Point const &point_2,
55             Point const &point_3)
56        : point_1_{point_1}, point_2_{point_2}, point_3_{point_3},
57          side_1_{point_1.length(point_2)}, side_2_{point_2.length(point_3)},
58          side_3_{point_3.length(point_1)} {}
59    // Dtor.
```

TP 4 - Problem 4 - IV

```
60      ~Triangle() {}
61      void print() const override {
62        std::cout << "Triangle{" << point_1_ << ' ' << point_2_ << ' '
63                  << point_3_ << "}\n";
64      }
65      double area() const override {
66        if (CheckIfTriangle()) {
67          // Using Heron's formula.
68          double const s{(side_1_ + side_2_ + side_3_) / 2};
69          return std::sqrt(s * (s - side_1_) * (s - side_2_) * (s - side_3_));
70        } else
71          return NAN;
72      }
73      double perimeter() const override {
74        return CheckIfTriangle() ? side_1_ + side_2_ + side_3_ : NAN;
75      }
76      void SetPoints(Point const &point_1, Point const &point_2,
77                     Point const &point_3) {
78        point_1_ = point_1;
79        point_2_ = point_2;
```

TP 4 - Problem 4 - V

```
80        point_3_ = point_3;
81        side_1_ = point_1.length(point_2);
82        side_2_ = point_2.length(point_3);
83        side_3_ = point_3.length(point_1);
84      }
85      double side_1() const { return side_1_; }
86      double side_2() const { return side_2_; }
87      double side_3() const { return side_3_; }
88
89    private:
90      bool CheckIfTriangle() const {
91        if (std::isnan(side_1_) || std::isnan(side_2_) || std::isnan(side_3_))
92          return false;
93        return (side_1_ + side_2_ > side_3_) &&
94               (side_1_ + side_3_ > side_2_) && (side_2_ + side_3_ > side_1_);
95      }
96      Point point_1_;
97      Point point_2_;
98      Point point_3_;
99      double side_1_;
```

TP 4 - Problem 4 - VI

```
100     double side_2_;
101     double side_3_;
102   };
103
104   class Rectangle : public Shape {
105   public:
106     // Default ctor.
107     Rectangle() : point_{}, width_{NAN}, height_{NAN} {}
108     // Parametrised ctor.
109     Rectangle(Point const &point, double width, double height)
110         : point_{point}, width_{width}, height_{height} {
111       if (width <= 0)
112         throw std::domain_error{"Rectangle: the width must be positive."};
113       if (height <= 0)
114         throw std::domain_error{"Rectangle: the height must be positive."};
115     }
116     // Dtor.
117     ~Rectangle() {}
118     void print() const override {
119       std::cout << "Rectangle{" << point_ << ' ' << width_ << ' ' << height_
```

TP 4 - Problem 4 - VII

```
120                 << "}\n";
121     }
122     double area() const override { return width_ * height_; }
123     double perimeter() const override { return 2 * (width_ + height_); }
124     void SetWidth(double width) {
125       if (width <= 0)
126         throw std::domain_error{
127             "Rectangle::SetWidth: the width must be positive."};
128       width_ = width;
129     }
130     void SetHeight(double height) {
131       if (height <= 0)
132         throw std::domain_error{
133             "Rectangle::SetHeight: the height must be positive."};
134       height_ = height;
135     }
136     Point GetCorner() const { return point_; }
137     double GetWidth() const { return width_; }
138     double GetHeight() const { return height_; }
139
```

TP 4 - Problem 4 - VIII

```
140   private:
141     Point const point_;
142     double width_;
143     double height_;
144   };
145
146   class Circle : public Shape {
147   public:
148     // Parametrised ctor.
149     Circle(Point const &point, double radius)
150         : point_{point}, radius_{radius} {
151       if (radius <= 0)
152         throw std::domain_error{"Circle: the radius must be positive."};
153     }
154     // Dtor.
155     ~Circle() {}
156     void print() const override {
157       std::cout << "Circle{" << point_ << ' ' << radius_ << "}\n";
158     }
159     double area() const override {
```

TP 4 - Problem 4 - IX

```
160       return std::numbers::pi * radius_ * radius_;
161     }
162     double perimeter() const override {
163       return 2 * std::numbers::pi * radius_;
164     }
165
166   private:
167     Point const point_;
168     double const radius_;
169   };
170
171   class GoodSquare : public Shape {
172   public:
173     // Parametrised ctor.
174     GoodSquare(Point const &point, double size) : rect_{point, size, size} {
175       if (size <= 0)
176         throw std::domain_error{"GoodSquare: the size must be positive."};
177     }
178     // Dtor.
179     ~GoodSquare() {}
```

TP 4 - Problem 4 - X

```
180     void print() const override {
181       std::cout << "GoodSquare{" << rect_.GetCorner() << ' '
182                 << rect_.GetWidth() << "}\n";
183     }
184     double area() const override {
185       return rect_.GetWidth() * rect_.GetWidth();
186     }
187     double perimeter() const override { return 4 * rect_.GetWidth(); }
188
189   private:
190     Rectangle rect_;
191   };
192   void characteristics(Shape const *v[], int n) {
193     std::cout << "\nC-style characteristics function output:\n";
194     for (int i{}; i < n; ++i) {
195       v[i]->print();
196       std::cout << "area: " << v[i]->area()
197                 << " perimeter: " << v[i]->perimeter() << '\n';
198     }
199   }
```

TP 4 - Problem 4 - XI

```
200  void characteristics(std::vector<Shape const *> const &v) {
201    std::cout << "\nSTL-style characteristics function output:\n";
202    for (auto shape_ptr : v) {
203      shape_ptr->print();
204      std::cout << "area: " << shape_ptr->area()
205                << " perimeter: " << shape_ptr->perimeter() << '\n';
206    }
207  }
208
209  int main() {
210    SHOW(Point{}.length(Point{}))
211    SHOW((Point{0, 0}.length(Point{0, 1})))
212    SHOW((Point{0, 0}.length(Point{1, 0})))
213    SHOW((Point{0, 0}.length(Point{1, 1})))
214    Triangle{}.print();
215    // Not a triangle.
216    SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{0, 2}}.perimeter()))
217    SHOW((Triangle{Point{0, 0}, Point{1, 1}, Point{2, 2}}.perimeter()))
218    // Triangle.
219    SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.perimeter()))
```

TP 4 - Problem 4 - XII

```
220     SHOW((Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.area()))
221     // Rectangle
222     Rectangle{}.print();
223     Rectangle{Point{1, 1}, 2, 3}.print();
224     SHOW((Rectangle{Point{1, 1}, 2, 3}.perimeter()))
225     SHOW((Rectangle{Point{1, 1}, 2, 3}.area()))
226     // Circle
227     Circle{Point{0, 0}, 1}.print();
228     SHOW((Circle{Point{0, 0}, 1}.area()));
229     SHOW((Circle{Point{0, 0}, 1}.perimeter()));
230     // GoodSquare
231     GoodSquare{Point{0, 0}, 2}.print();
232     SHOW((GoodSquare{Point{0, 0}, 2}.area()));
233     SHOW((GoodSquare{Point{0, 0}, 2}.perimeter()));
234
235     Triangle my_triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}};
236     Rectangle my_rectangle{Point{1, 1}, 2, 3};
237     Circle my_circle{Point{0, 0}, 1};
238     GoodSquare my_square{Point{0, 0}, 2};
239     {
```

TP 4 - Problem 4 - XIII

```
240       // C style array.
241       Shape const *my_menagerie[]{&my_triangle, &my_rectangle, &my_circle,
242                                   &my_square};
243       int constexpr my_menagerie_sz{sizeof my_menagerie /
244                                     sizeof my_menagerie[0]};
245       characteristics(my_menagerie, my_menagerie_sz);
246     }
247     {
248       // STL style array.
249       std::vector<Shape const *> my_menagerie{&my_triangle, &my_rectangle,
250                                               &my_circle, &my_square};
251       characteristics(my_menagerie);
252     }
253     return 0;
254   }
```

TP 4 - Problem 4 - XIV

Output:

```
1  Macro SHOW “Point{}.length(Point{})”: nan
2  Macro SHOW “(Point{0, 0}.length(Point{0, 1}))”: 1
3  Macro SHOW “(Point{0, 0}.length(Point{1, 0}))”: 1
4  Macro SHOW “(Point{0, 0}.length(Point{1, 1}))”: 1.41421
5  Triangle{(nan nan) (nan nan) (nan nan)}
6  Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{0,
    ↪  2}}.perimeter())”: nan
7  Macro SHOW “(Triangle{Point{0, 0}, Point{1, 1}, Point{2,
    ↪  2}}.perimeter())”: nan
8  Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{1,
    ↪  0}}.perimeter())”: 3.41421
9  Macro SHOW “(Triangle{Point{0, 0}, Point{0, 1}, Point{1, 0}}.area())”:
    ↪  0.5
10 Rectangle{(nan nan) nan nan}
11 Rectangle{(1 1) 2 3}
12 Macro SHOW “(Rectangle{Point{1, 1}, 2, 3}.perimeter())”: 10
13 Macro SHOW “(Rectangle{Point{1, 1}, 2, 3}.area())”: 6
14 Circle{(0 0) 1}
```

TP 4 - Problem 4 - XV

```
15  Macro SHOW "(Circle{Point{0, 0}, 1}.area())": 3.14159
16  Macro SHOW "(Circle{Point{0, 0}, 1}.perimeter())": 6.28319
17  GoodSquare{(0 0) 2}
18  Macro SHOW "(GoodSquare{Point{0, 0}, 2}.area())": 4
19  Macro SHOW "(GoodSquare{Point{0, 0}, 2}.perimeter())": 8
20
21  C-style characteristics function output:
22  Triangle{(0 0) (0 1) (1 0)}
23  area: 0.5 perimeter: 3.41421
24  Rectangle{(1 1) 2 3}
25  area: 6 perimeter: 10
26  Circle{(0 0) 1}
27  area: 3.14159 perimeter: 6.28319
28  GoodSquare{(0 0) 2}
29  area: 4 perimeter: 8
30
31  STL-style characteristics function output:
32  Triangle{(0 0) (0 1) (1 0)}
33  area: 0.5 perimeter: 3.41421
34  Rectangle{(1 1) 2 3}
```

TP 4 - Problem 4 - XVI

```
35  area: 6 perimeter: 10
36  Circle{(0 0) 1}
37  area: 3.14159 perimeter: 6.28319
38  GoodSquare{(0 0) 2}
39  area: 4 perimeter: 8
```