TP 3 - Problem 2 - I

```cpp
#include <cassert>
#include <cmath>
#include <iostream>
#include <numbers>

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

// A class to represent a point identified by his cartesian coordinates.
class CartesianPoint {
public:
  // Default ctor. The data member are initialised to zero.
  CartesianPoint() : x_{}, y_{} {}
  // Copy ctor.
  CartesianPoint(CartesianPoint const &other)
      : x_{other.x_}, y_{other.y_} {}
  // Another ctor.
  CartesianPoint(double x, double y) : x_{x}, y_{y} {}
  // Getters.
  double get_x() const { return x_; }
```

TP 3 - Problem 2 - II

```
20    double get_y() const { return y_; }
21    // Setters.
22    void set_x(double x) { x_ = x; }
23    void set_y(double y) { y_ = y; }
24    // Returns the distance between two points.
25    double distanceTo(CartesianPoint const &other) const {
26      return std::sqrt((x_ - other.x_) * (x_ - other.x_) +
27                       (y_ - other.y_) * (y_ - other.y_));
28    }
29
30  private:
31    double x_;
32    double y_;
33  };
34
35  // A class to represent a point identified by his polar coordinates.
36  class PolarPoint {
37  public:
38    // Default ctor.
39    PolarPoint() : r_{}, theta_{} {}
```

TP 3 - Problem 2 - III

```
40      // Copy ctor.
41      PolarPoint(PolarPoint const &other)
42          : r_{other.r_}, theta_{other.theta_} {}
43      // Another ctor.
44      PolarPoint(double r, double theta) : r_{r}, theta_{theta} {}
45      // Getters.
46      double get_r() const { return r_; }
47      double get_theta() const { return theta_; }
48      // Setters.
49      void set_r(double r) { r_ = r; }
50      void set_theta(double theta) { theta_ = theta; }
51
52    private:
53      double r_;
54      double theta_;
55    };
56
57    // Conversion free function.
58    CartesianPoint polarToCartesian(PolarPoint const &p) {
59      return {p.get_r() * std::cos(p.get_theta()),
```

TP 3 - Problem 2 - IV

```
60              p.get_r() * std::sin(p.get_theta())};
61  }
62
63  int main() {
64    CartesianPoint o;
65    SHOW(o.get_x())
66    SHOW(o.distanceTo(o))
67    CartesianPoint p{0, 1};
68    SHOW(p.get_y())
69    SHOW(o.distanceTo(p))
70    SHOW(p.distanceTo(o))
71    p.set_y(0);
72    SHOW(p.get_y())
73    SHOW(o.distanceTo(p))
74    SHOW(polarToCartesian(PolarPoint{1, 0}).get_x())
75    SHOW(polarToCartesian(PolarPoint{1, 0}).get_y())
76    SHOW(polarToCartesian(PolarPoint{1, std::numbers::pi / 4}).get_x())
77    SHOW(polarToCartesian(PolarPoint{1, std::numbers::pi / 4}).get_y())
78    SHOW(polarToCartesian(PolarPoint{1, std::numbers::pi / 4}).distanceTo(o))
```

TP 3 - Problem 2 - V

```
79      return 0;
80  }
```

Output:

```
1   Macro SHOW “o.get_x()”: 0
2   Macro SHOW “o.distanceTo(o)”: 0
3   Macro SHOW “p.get_y()”: 1
4   Macro SHOW “o.distanceTo(p)”: 1
5   Macro SHOW “p.distanceTo(o)”: 1
6   Macro SHOW “p.get_y()”: 0
7   Macro SHOW “o.distanceTo(p)”: 0
8   Macro SHOW “polarToCartesian(PolarPoint{1, 0}).get_x()”: 1
9   Macro SHOW “polarToCartesian(PolarPoint{1, 0}).get_y()”: 0
10  Macro SHOW “polarToCartesian(PolarPoint{1, std::numbers::pi /
     ↪  4}).get_x()”: 0.707107
11  Macro SHOW “polarToCartesian(PolarPoint{1, std::numbers::pi /
     ↪  4}).get_y()”: 0.707107
12  Macro SHOW “polarToCartesian(PolarPoint{1, std::numbers::pi /
     ↪  4}).distanceTo(o)”: 1
```