TP 3 - Problem 3 - I

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

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

class Polynomial {
public:
  // Default ctor.
  Polynomial() : coeffs_{} {}
  // Parametrized ctor with the vector of coefficients.
  Polynomial(std::vector<double> const &coeffs) : coeffs_{coeffs} {}
  // Another parametrized ctor with the constant coefficient.
  Polynomial(double a_0) : coeffs_{a_0} {}
  // Evaluate the polynomial at x using the Horner algorithm.
  double evaluate(double const x) const {
    if (coeffs_.empty())
      return 0;
    // The function member “size” returns an unsigned int...
    size_t i{coeffs_.size()};
```

TP 3 - Problem 3 - II

```
20      // Be careful not to decrement an unsigned integer equal to 0. Then, we
21      // iterate from n to 1 and use i-1 as index.
22      double result{coeffs_.at(i - 1)};
23      while (--i > 0)
24        result = result * x + coeffs_.at(i - 1);
25      return result;
26    }
27    Polynomial &add(Polynomial const &other) {
28      // This is empty: copy the other.
29      if (coeffs_.empty())
30        coeffs_ = other.coeffs_;
31      // The other is empty: no op.
32      else if (other.coeffs_.empty())
33        /* nothing */;
34      else {
35        size_t max_sz{std::max(coeffs_.size(), other.coeffs_.size())};
36        for (size_t i{}; i < max_sz; ++i) {
37          // The other is larger than the this.
38          if (i >= coeffs_.size())
39            coeffs_.push_back(other.coeffs_.at(i));
```

TP 3 - Problem 3 - III

```
40            // The this is larger than the other.
41            else if (i >= other.coeffs_.size())
42              /* nothing */;
43            else
44              coeffs_.at(i) += other.coeffs_.at(i);
45          }
46        }
47        // Returns a reference on this: you can code
48        // “poly.add(Polynomial{1}).evaluate(3)”.
49        return *this;
50      }
51      // Returns a external representation of the polynomial.
52      std::string to_string() {
53        std::string str{"["};
54        for (auto const &a_i : coeffs_)
55          str += std::to_string(a_i) + ' ';
56        // The last space is crushed.
57        str.back() = ']';
58        return str;
59      }
```

TP 3 - Problem 3 - IV

```
60
61  private:
62    // The coefficients, stored as [a_0, a_1, ..., a_n],  when the polynomial
63    // is a_0 + a_1 x + a_2 x^2 +  ... + a_n x^n.
64    std::vector<double> coeffs_;
65  };
66
67  int main() {
68    Polynomial polynomial{{1, 2, 3}};
69    SHOW(polynomial.to_string())
70    SHOW(polynomial.evaluate(2));
71    {
72      Polynomial polynomial{9};
73      SHOW(polynomial.to_string())
74    }
75    {
76      // This is the largest.
77      Polynomial polynomial{{1, 2, 3}};
78      SHOW(polynomial.add(Polynomial{{1, 2}}).to_string())
79    }
```

TP 3 - Problem 3 - V

```
80      {
81        // Other is the largest.
82        Polynomial polynomial{{1, 2, 3}};
83        SHOW(polynomial.add(Polynomial{{1, 2, 3}}).to_string())
84      }
85      {
86        // Same size.
87        Polynomial polynomial{{1, 2, 3}};
88        SHOW(polynomial.add(Polynomial{{1, 2, 3, 4}}).to_string())
89      }
90      {
91        // This is empty.
92        Polynomial polynomial;
93        SHOW(polynomial.add(Polynomial{{1, 2, 3}}).to_string())
94      }
95      {
96        // Other is empty.
97        Polynomial polynomial{{1, 2, 3}};
98        SHOW(polynomial.add(Polynomial{}).to_string())
99      }
```

TP 3 - Problem 3 - VI

```
100     {
101       // Both are the same.
102       Polynomial polynomial{{1, 2, 3}};
103       SHOW(polynomial.add(polynomial).to_string())
104     }
105     return 0;
106   }
```

TP 3 - Problem 3 - VII

Output:

```
1  Macro SHOW “polynomial.to_string()”: [1.000000 2.000000 3.000000]
2  Macro SHOW “polynomial.evaluate(2)”: 17
3  Macro SHOW “polynomial.to_string()”: [9.000000]
4  Macro SHOW “polynomial.add(Polynomial{{1, 2}}).to_string()”: [2.000000
    ↪  4.000000 3.000000]
5  Macro SHOW “polynomial.add(Polynomial{{1, 2, 3}}).to_string()”:
    ↪  [2.000000 4.000000 6.000000]
6  Macro SHOW “polynomial.add(Polynomial{{1, 2, 3, 4}}).to_string()”:
    ↪  [2.000000 4.000000 6.000000 4.000000]
7  Macro SHOW “polynomial.add(Polynomial{{1, 2, 3}}).to_string()”:
    ↪  [1.000000 2.000000 3.000000]
8  Macro SHOW “polynomial.add(Polynomial{}).to_string()”: [1.000000
    ↪  2.000000 3.000000]
9  Macro SHOW “polynomial.add(polynomial).to_string()”: [2.000000 4.000000
    ↪  6.000000]
```