TP 3 - Problem 3 - I

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

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

class Polynomial {
public:
  // Default ctor: set a degree 0 polynomial with a_0 = 0.
  Polynomial() : coeffs_{0} {}
  // Parametrized ctor with the vector of coefficients.
  Polynomial(std::vector<double> const &coeffs) : coeffs_{coeffs} {
    // The standard way to report errors that arise because an argument
    // value has not been accepted.
    if (coeffs.empty())
      throw std::invalid_argument{"Attempt to construct a polynomial with "
                                  "an empty vector of coefficients"};
  }
  // Another parametrized ctor with the constant coefficient.
```

TP 3 - Problem 3 - II

```
20      Polynomial(double a_0) : coeffs_{a_0} {}
21      // Evaluate the polynomial at x using the Horner algorithm.
22      double evaluate(double const x) const {
23        // The function member “size” returns an unsigned int...
24        size_t i{coeffs_.size()};
25        // Be careful not to decrement an unsigned integer equal to 0. Then, we
26        // iterate from n to 1 and use i-1 as index.
27        double result{coeffs_.back()};
28        while (--i > 0)
29          result = result * x + coeffs_.at(i - 1);
30        return result;
31      }
32      // Returns a new instance of “Polynomial”.
33      Polynomial add(Polynomial const &other) {
34        size_t max_sz{std::max(coeffs_.size(), other.coeffs_.size())};
35        std::vector<double> coeffs;
36        // As the size is known, it is smart to reserve capacity to avoid
37        // reallocations.
38        coeffs.reserve(max_sz);
39        for (size_t i{}; i < max_sz; ++i) {
```

TP 3 - Problem 3 - III

```
40        // Other is larger than this: push at end only other coefficient.
41        if (i >= coeffs_.size())
42          coeffs.push_back(other.coeffs_.at(i));
43        // This is larger than other: push at end only this coefficent.
44        else if (i >= other.coeffs_.size())
45          coeffs.push_back(coeffs_.at(i));
46        else
47          // Push at end the sum of this coefficient and other coefficient.
48          coeffs.push_back(coeffs_.at(i) + other.coeffs_.at(i));
49      }
50      return {coeffs};
51    }
52    // Returns a external representation of the polynomial.
53    std::string to_string() const {
54      std::string str{"["};
55      for (auto const &a_i : coeffs_)
56        str += std::to_string(a_i) + ' ';
57      // The last space is crushed.
58      str.back() = ']';
59      return str;
```

TP 3 - Problem 3 - IV

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

TP 3 - Problem 3 - V

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

TP 3 - Problem 3 - VI

```
100     }
101     {
102       // Both are the same.
103       Polynomial polynomial{{1, 2, 3}};
104       SHOW(polynomial.add(polynomial).to_string())
105     }
106     return 0;
107   }
```

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]
```