# TP 1 - Partie 2 - Exercice 7 - I

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

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

// Returns poly[0] + poly[1]*x^1 + ... + poly[n-1]*x^(n-1)
double horner_recursive_eval(double const poly[], int n, double x) {
  assert(n > 0);
  if (n == 1)
    return poly[0];
  return poly[0] + x * horner_recursive_eval(poly + 1, n - 1, x);
}
// Returns poly[0] + poly[1]*x^1 + ... + poly[n-1]*x^(n-1)
double horner_iterative_eval(double const poly[], int n, double x) {
  assert(n > 0);
  --n;
  double result{poly[n]};
```

# TP 1 - Partie 2 - Exercice 7 - II

```
20      while (--n >= 0)
21        result = result * x + poly[n];
22      return result;
23    }
24    bool unit_test() {
25      {
26        double constexpr poly[]{1, 2, 3};
27        double const result{poly[0] + poly[1] * 3 + poly[2] * 3 * 3};
28        assert(horner_recursive_eval(poly, 3, 3) == result);
29        assert(horner_iterative_eval(poly, 3, 3) == result);
30      }
31      {
32        double constexpr poly[]{3, 2, 1};
33        double const result{poly[0] + poly[1] * 3 + poly[2] * 3 * 3};
34        assert(horner_recursive_eval(poly, 3, 3) == result);
35        assert(horner_iterative_eval(poly, 3, 3) == result);
36      }
37      {
38        double constexpr poly[]{-1, -2, -3};
39        double const result{poly[0] + poly[1] * 3 + poly[2] * 3 * 3};
```

# TP 1 - Partie 2 - Exercice 7 - III

```
40        assert(horner_recursive_eval(poly, 3, 3) == result);
41        assert(horner_iterative_eval(poly, 3, 3) == result);
42      }
43      {
44        double constexpr poly[]{-1, 0, 1};
45        double const result{poly[0] + poly[1] * 3 + poly[2] * 3 * 3};
46        assert(horner_recursive_eval(poly, 3, 3) == result);
47        assert(horner_iterative_eval(poly, 3, 3) == result);
48      }
49      return true;
50    }
51    double normcdf(double x) {
52      if (x < 0)
53        return 1 - normcdf(-x);
54      // The a_0 is equal to 0.
55      double constexpr poly[]{0,           0.319381530,  -0.356563782,
56                              1.781477937, -1.821255978, 1.330274429};
57      int constexpr n{sizeof poly / sizeof poly[0]};
58      double const k{1 / (1 + 0.2316419 * x)};
59      return 1 - 1 / std::sqrt(2 * std::numbers::pi) * std::exp(-x * x / 2) *
```

# TP 1 - Partie 2 - Exercice 7 - IV

```
60                    horner_iterative_eval(poly, n, k);
61  }
62  int main() {
63    assert(unit_test());
64    std::cout << "0.0228 expected - ", SHOW(normcdf(-2));
65    std::cout << "0.1587 expected - ", SHOW(normcdf(-1));
66    std::cout << "0.5 expected - ", SHOW(normcdf(0));
67    std::cout << "0.8413 expected - ", SHOW(normcdf(1));
68    std::cout << "0.9772 expected - ", SHOW(normcdf(2));
69  }
```

Output:

```
1  0.0228 expected - Macro SHOW “normcdf(-2)”: 0.0227501
2  0.1587 expected - Macro SHOW “normcdf(-1)”: 0.158655
3  0.5 expected - Macro SHOW “normcdf(0)”: 0.5
4  0.8413 expected - Macro SHOW “normcdf(1)”: 0.841345
5  0.9772 expected - Macro SHOW “normcdf(2)”: 0.97725
```