```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;


// ################ QUESTION 1 ################
// ------------------------------------------------------------------------
// Classe "FUNCTION" : classe de base
class Function {
public:
  virtual double evaluate(double x) const;
  virtual void display() const;

  ~Function() {};
};

double Function::evaluate(double x) const {}

void Function::display() const {}



// ----------
// Classe "GAUSSIAN" : classe dérivée de "Function"
class Gaussian : public Function {
private:
  double A, B, C, D;

public:
  Gaussian(double A, double B, double C, double D) :
    A(A), B(B), C(C), D(D) {}
  ~Gaussian() {};

  virtual double evaluate(double x) const override;
  virtual void display() const override;
};

// Calcul de la valeur de la fonction gaussienne
double Gaussian::evaluate(double x) const {
  return A * exp(-(pow(x - C, 2) / pow(B, 2)) + D);
}

// Affiche la fonction gaussienne
void Gaussian::display() const {
  cout << "f(x) = " << A << " * exp(-((x - " << C << ")^2) / (" << B << "^2)) + " << D << endl;
}




// ################ QUESTION 2 ################
// ------------------------------------------------------------------------
class Term : public Function {
public:
  // Coefficient et exposant de x
  double coefficient;
  int exponent;

  // Constructeur
  Term(double coefficient, int exponent) : coefficient(coefficient), exponent(exponent) {}

};
```

```
// ----------
// Classe "POLYNOMIAL" : classe dérivée de "Term"
class Polynomial : public Function {
private:
  vector<Term> terms;

public:
  Polynomial();

  virtual double evaluate(double x) const override;
  virtual void display() const override;

  virtual void AddTerm(double coefficient, int exponent);
};


// Ajoute un terme au polynôme
void Polynomial::AddTerm(double coefficient, int exponent)
{
  for(int i = 0; i < terms.size(); i++)
  {
    if(terms[i].exponent == exponent)
    {
      terms[i].coefficient = coefficient;
      cout << "Coefficient remplacé : " << coefficient << endl;
      return;
    }
  }
  terms.emplace_back(coefficient, exponent);
  cout << "Terme ajouté : " << coefficient << " * x^" << exponent << endl;
}

// Calcul de la valeur du polynôme
double Polynomial::evaluate(double x) const {

  double result = 0;

  for (int i = 0; i < terms.size(); i++) {
    result += terms[i].coefficient * pow(x, terms[i].exponent);
  }

  return result;
}

// Affiche le polynôme
void Polynomial::display() const {

  cout << "f(x) = ";
  for (int i = 0; i < terms.size(); i++) {
    cout << terms[i].coefficient << " * x^" << terms[i].exponent << " + ";
  }
  cout << endl;

}









// ################ FONCTION MAIN ################
// --------------------------------------------------------------------
```

```
// Fonction principale
int main()
{
  // Test de la classe "Gaussian"
  Gaussian g(2, 3, 1.5, 7);
  double result = g.evaluate(8.5);


  g.display();
  cout << "f(8.5) = " << result << endl;



  // Test de la classe "Term"
  Polynomial p;
  p.AddTerm(4, 7);

  double result_poly = p.evaluate(8.5);
  p.display();
}
```