1 nombre.hpp

```
#pragma once

#include <string>
#include <cassert>
#include <cmath>

class nombre {
private:
  const int maxBits = sizeof(long long int)*8;
  long long int n;
  // méthodes auxiliaires
  static long long int horner(const std::string &s);
  std::string toBin(const int nbBits) const ;
  void toDec(const std::string &s);

public:
  // les constructeurs
  nombre(const long long int x) : n(x) {}
  nombre(const std::string &b) {
    this->toDec(b);
  }
  // accesseurs et modicateurs
  long long int getDécimal() const ;
  std::string getBinaire(const int nbBits) const;
  void setDécimal(const long long int n);
  void setBinaire(const std::string &b);
  //
  // les opérateurs
  nombre operator+(const nombre &n) const;
  nombre operator-(const nombre &n) const;
  nombre operator*(const nombre &n) const;
  nombre operator/(const nombre &n) const;
  nombre operator%(const nombre &n) const;
};
```

2 nombre.cpp

```
#include <string>
#include <cassert>
#include <cmath>
#include <iostream>

#include "nombre.hpp"

/**
 * Rôle : renvoie la forme décimal de this sur nbBits
 */
long long int nombre::getDécimal() const {
  return this->n;
}

/**
 * Rôle : renvoie la forme binaire de this sur nbBits
 */
std::string nombre::getBinaire(const int nbBits) const {
  return this->toBin(nbBits);
}
```



```
}

/**
 * Rôle : affecte l'entier décimal n au nombre courant
 */
void nombre::setDécimal(const long long int n) {
  this->n = n;
}

/**
 * Rôle : affecte l'entier binaire n au nombre courant
 */
void nombre::setBinaire(const std::string &b) {
  this->toDec(b);
}

/**
 * Antécédent : b une chaîne de caractère représentant un nombre binaire valide
 * Rôle       : renvoie sa conversion sous forme d'un entier
 * Note       : algorithme de Horner
 */
long long int nombre::horner(const std::string &b) {
  int base=2, size = b.length();
  long long int n=0;
  for (int i=0; i<size; i++)
    n = n*base + b[i]-'0';
    // n = \sum_{k=size}^{i} b[k]x^{k-i}
  // n = \sum_{k=size}^{0} b[k]x^{k}
  return n;
}

/**
 * Antécédent : b une chaîne de caractère représentant un nombre binaire valide
 *              en complément à 2
 * Rôle       : renvoie sa conversion sous forme d'un entier
 */
void nombre::toDec(const std::string &b) {
  if (b[0] == '0')
    this->n = horner(b);
  else
    // le chiffre est un 1 => nombre négatif
    this->n  = horner(b) - (1LL<<b.length());
}

/**
 * Antécédent : nbBits <= this->maxBits
 * Rôle : cette méthode renvoie la représentation binaire sur nbBits
 *        sous forme d'une std::string du nombre décimal courant
 */
std::string nombre::toBin(const int nbBits) const {
  assert(nbBits<=this->maxBits);
  std::string s="";
  if (this->n==0)
    return s.insert(0, nbBits, '0');
  // calculer le plus gd entier sur nbBits
  const long long int nbBitsP2 = 1LL<<nbBits;
  long long int _n = this->n;
  // vérifier que _n est dans les bornes
```

```
  // et prendre son complément à 2 s'il est négatif
  if (_n<0) {
      assert(_n>=-nbBitsP2/2);
      // prendre le complément de n à
      _n+=nbBitsP2;
  }
  else
     assert(_n<nbBitsP2/2);
  // ok = n est dans les bornes => le décomposer en base 2
  // dans la chaîne de caractères s
  do
    s =  std::to_string(_n%2) + s;
  while ((_n/=2)!=0);
  // compléter avec les bits de poids fort à 0
  return s.insert(0, nbBits-s.size(), '0');
}

nombre nombre::operator+(const nombre &n) const { return nombre(this->n + n.getDécimal
nombre nombre::operator-(const nombre &n) const { return nombre(this->n - n.getDécimal
nombre nombre::operator*(const nombre &n) const { return nombre(this->n * n.getDécimal
nombre nombre::operator%(const nombre &n) const { return nombre(this->n % n.getDécimal
nombre nombre::operator/(const nombre &n) const {
  assert(n.getDécimal()!=0);
  return nombre(this->n / n.getDécimal());
}
```