```cpp
#include <string>
#include <iostream>
#include "bigint.hpp"

// Les constructeurs

/*
 * Antécédent : n>=0
 *
 * Rôle : initialise le BigInt courant à partir d'un
 *        int qui contient les chiffres du nombre
 *
 * Note : les chiffres de poids faible sont placés en tête
 */
BigInt::BigInt(unsigned int n) {
  std::string s= "";
  do {
    s += std::to_string(n % BigInt::base);
    n /= BigInt::base;
  } while (n!=0);
  this->leschiffres = s;
}

/*
 * Antécédent : s représente un entier naturel valide
 *
 * Rôle : initialise le BigInt courant à partir d'une
 *        string qui contient les chiffres du nombre
 *
 * Note : les chiffres de poids faible sont placés en tête
 */
BigInt::BigInt(const std::string s) {
  for (char c : s)
    this->leschiffres = c + this->leschiffres;
}

// Les méthodes

/*
 * Rôle : renvoie le nombre de chiffres contenus dans le BigInt courant
 */
int BigInt::size() const {
  return this->leschiffres.length();
}

/*
 * Rôle : renvoie la somme *this+n
 */
BigInt BigInt::add(const BigInt& n) const {
  BigInt xmin, xmax;
  // déterminer le bigInt qui possède le plus de chiffres
  // et celui qui en possède le moins
  if (this->size()< n.size()) {
    xmin=*this; xmax=n;
  } else {
    xmin=n; xmax=*this;
  }
  // faire l'addition des chiffres
  std::string s="";
  int ret=0;
  for (int i=0; i<xmax.size(); i++) {
    // ajouter le chiffre courant de xmax + la retenue
    int c = xmax.leschiffres[i]-'0'+ret;
    // s'il existe, ajouter le chiffre courant de xmin
    if (i<xmin.size()) c+=xmin.leschiffres[i]-'0';
    // la somme des chiffres provoque-t-elle un retenue ?
```



```cpp
    if (c<BigInt::base) ret=0;
    else { // il y a une retenue
      ret=1; c-=BigInt::base;
    }
    //
    s = std::to_string(c) + s;
  }
  // une dernière retenue ?
  if (ret==1) s = "1"+ s;
  return BigInt(s);
}

// Les surcharges

/*
 * Rôle : surcharge de l'opérateur +
 *        renvoie la somme *this+n
 */
BigInt BigInt::operator+(const BigInt &b) const {
  return this->add(b);
}

/*
 * Rôle : surcharge de l'opérateur <<
 *        écrit le BigInt b sur le flot f
 */
std::ostream& operator<< (std::ostream& f, const BigInt& b)  {
  for (int i = b.size()-1; i>=0; i--)
    f<< b.leschiffres[i];
  return f;
}
```

```hpp
#ifndef BIGINT_HPP
#define BIGINT_HPP

#include <iostream>
#include <string>

class BigInt {
private:
  // Note : les chiffres de poids faible sont placés en tête
  std::string leschiffres;
public:
  static const int base = 10;
  // constructeurs
  BigInt(const std::string s="0");
  BigInt(unsigned int n);
  // les méthodes
  int size() const; // nombre de chiffres du BigInt courant
  BigInt add(const BigInt& n) const;
  // les surcharges
  BigInt operator+(const BigInt &b) const;
  friend std::ostream &operator<<(std::ostream &f, const BigInt& b);
};

#endif
```



```cpp
#include <cstdlib>
#include <string>
#include <iostream>

#include "bigint.hpp"

int main() {

  BigInt n1 = BigInt(2);
  BigInt n2 = BigInt("99999999999999999999999999999999999999999999999999");

  std::cout << n1 << std::endl;
  std::cout << n2 << std::endl;
  std::cout << n2.size() << std::endl;
  std::cout << (n1 + n2) << std::endl;
}
```