```cpp
/*
-------------------------------------------------------------------------------
Nom du fichier  : main.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 10.03.2022
Description     : Fichier principal du programme permettant de tester le bon
                  fonctionnement de la classe Matrix.
Compilateur     : Mingw-w64 g++ 8.1.0
-------------------------------------------------------------------------------
*/
#include <iostream>
#include "matrix.hpp"

void test();

int main(int argc, char *argv[]) {
    // Décommentez la ligne ci-dessous pour run les tests
    // test();

    // Partie où l'on test que les arguments passés en lignes de commandes sont
    // cohérents
    if(argc != 6){
        throw std::runtime_error("Pas le bon nombre d'argument.");
    }

     // Si l'un des arguments ne peut pas être transformé en int, le programme crash
     int nbRow1 = std::stoi(argv[1]);
     int nbCol1 = std::stoi(argv[2]);
     int nbRow2 = std::stoi(argv[3]);
     int nbCol2 = std::stoi(argv[4]);
     int modulus = std::stoi(argv[5]);

    if(nbRow1 < 0 || nbRow2 < 0 || nbCol1 < 0 || nbCol2 < 0){
        throw std::runtime_error("Les nombres de lignes et de colonnes dovent etre "
                                            "des entiers positifs ou nuls.");
    }
     if(modulus < 1){
         throw std::runtime_error("Le modulo doit etre un entier plus grand que 0.");
     }
     // Nous avons cette instruction afin d'éviter que le random soit toujours le
     // même dans la création des valeurs des matrices.
    srand ((unsigned int) time(NULL));
    Matrix matrix1((unsigned int) nbRow1,(unsigned int) nbCol1,
                        (unsigned int) modulus);
    Matrix matrix2((unsigned int) nbRow2,(unsigned int) nbCol2,
                        (unsigned int) modulus);
    std::cout << "The modulus is " << modulus << std::endl;

    std::cout << "One" << std::endl;
    std::cout << matrix1 << std::endl;

    std::cout << "Two" << std::endl;
    std::cout << matrix2 << std::endl;

    std::cout << "One + Two" << std::endl;
    std::cout << matrix1.addStaticNew(matrix2) << std::endl;

    std::cout << "One - Two" << std::endl;
    std::cout << matrix1.subStaticNew(matrix2) << std::endl;

    std::cout << "One * Two" << std::endl;
    std::cout << matrix1.multStaticNew(matrix2) << std::endl;

    test();
    return 0;
}
```

```cpp
/**
 * Fonction permettant de tester tous les fonctionnalités d'une Matrix
 * Il est important de noter que certains test sont commentés car il font crash le
 * programme. N'hésitez pas à les décommenter si vous souhaitez les tester.
 */
void test(){
    std::cout << "Creation d'une matrice vide" << std::endl;
    Matrix emptyMatrix(0, 0 ,5);
    std::cout << "Matrice vide :" << std::endl << emptyMatrix << std::endl;

    Matrix noRowMatrix(0, 2 ,5);
    std::cout << "Matrice sans ligne :" << std::endl << noRowMatrix << std::endl;

    Matrix noColMatrix(0, 0 ,5);
    std::cout << "Matrice sans colonne :" << std::endl << noColMatrix << std::endl;

    std::cout << "Creation d'une matrice avec un modulo valant 0" << std::endl;

    try{
        Matrix zeroModuloMat(2, 3 ,0);
    } catch(const std::runtime_error& e){
        std::cout << e.what() << std::endl << std::endl;
    }

    std::cout << "Creation de la matrice \"une\" de taille 2x3 avec 5 comme "
                 "modulo" << std::endl;
    Matrix one(2, 3 ,5);
    std::cout << "Matrice une :" << std::endl << one << std::endl;

    std::cout << "Creation de la matrice \"deux\" de taille 3x3 avec 5 comme "
                 "modulo" << std::endl;
    Matrix two(3, 3 ,5);
    std::cout << "Matrice deux :" << std::endl << two << std::endl;

    std::cout << "Creation de la matrice \"trois\" de taille 2x3 avec 7 comme "
                 "modulo" << std::endl;
    Matrix three(2, 3 ,7);
    std::cout << "Matrice trois :" << std::endl << three << std::endl;

    std::cout << "Creation de la matrice \"quatre\" copiant la matrice \"une\""<<
                 std::endl;
    Matrix four(one);
    std::cout << "Matrice quatre :" << std::endl << four << std::endl;

    std::cout << "Addition des matrices \"une\" et \"deux\"" << std::endl;
    std::cout << "Addition retournant par valeurs" << std::endl <<
    one.addStaticNew(two) << std::endl;

    Matrix* oneAddDynamic = one.addDynamicNew(two);
    std::cout << "Addition retournant par pointeur" << std::endl <<
    *oneAddDynamic << std::endl;
    std::cout << "Addition modifiant la premiere matrice" << std::endl;
    one.addItself(two);
    std::cout << one << std::endl << std::endl;

    std::cout << "Re-initialisation de la matrice \"une\" de taille 2x3 avec 5 comme "
                 "modulo" << std::endl;
    one = Matrix(2, 3 ,5);
    std::cout << "Matrice une :" << std::endl << one << std::endl;
    std::cout << "Matrice deux :" << std::endl << two << std::endl;

    std::cout << "Soustraction des matrices \"une\" et \"deux\" (une - deux)" <<
    std::endl;
    std::cout << "Soustraction retournant par valeurs" << std::endl <<
                 one.subStaticNew(two) << std::endl;
```

```
    Matrix* oneSubDynamic = one.subDynamicNew(two);
    std::cout << "Soustraction retournant par pointeur" << std::endl <<
                 *oneSubDynamic << std::endl;
    std::cout << "Soustraction modifiant la premiere matrice" << std::endl;
    one.subItself(two);
    std::cout << one << std::endl << std::endl;

    std::cout << "Re-initialisation de la matrice \"une\" de taille 2x3 avec 5 comme "
                     "modulo" << std::endl;
    one = Matrix(2, 3 ,5);
    std::cout << "Matrice une :" << std::endl << one << std::endl;
    std::cout << "Matrice deux :" << std::endl << two << std::endl;

    std::cout << "Multiplication des matrices \"une\" et \"deux\" (une * deux)" <<
                 std::endl;
    std::cout << "Multiplication retournant par valeurs" << std::endl <<
                 one.multStaticNew(two) << std::endl;

    Matrix* oneMultDynamic = one.multDynamicNew(two);
    std::cout << "Multiplication retournant par pointeur" << std::endl <<
                 *oneMultDynamic << std::endl;
    std::cout << "Multiplication modifiant la premiere matrice" << std::endl;
    one.multItself(two);
    std::cout << one << std::endl << std::endl;

    std::cout << "Re-initialisation de la matrice \"une\" de taille 2x3 avec 5 comme "
                     "modulo" << std::endl;
    one = Matrix(2, 3 ,5);
    std::cout << "Matrice une :" << std::endl << one << std::endl;

    std::cout << "Operation sur des matrices ayant differents modulos" << std::endl;
    std::cout << "\"une\" + \"trois\"" << std::endl;
    try{
        one.addStaticNew(three);
    } catch(const std::invalid_argument& e){
         std::cout << e.what() << std::endl;
    }
    std::cout << "\"une\" - \"trois\"" << std::endl;
    try{
        one.subDynamicNew(three);
    } catch(const std::invalid_argument& e){
         std::cout << e.what() << std::endl;
    }
    std::cout << "\"une\" * \"trois\"" << std::endl;
    try{
        one.multItself(three);
    } catch(const std::invalid_argument& e){
        std::cout << e.what() << std::endl;
    }
    delete oneSubDynamic;
    delete oneAddDynamic;
    delete oneMultDynamic;
}
```

```cpp
/*
-------------------------------------------------------------------------------
Nom du fichier  : matrix.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Classe permettant de modéliser des matrices de tailles diverses.
                  Il est possible de leur appliquer plusieurs opérations:
                  (L'addition, la soustraction ainsi que la multiplication).
                  Chaque opération peut se faire de 3 manières. Soit en modifiant
                  la matrice sur laquelle l'opération est invoquée. Soit en
                  renvoyant par valeur une nouvelle matrice allouée statiquement.
                  Soit en retournant un pointeur sur un nouvelle matrice allouée
                  dynamiquement.
                  Il est également possible d'afficher une matrice dans la console
                  via l'opérateur d'écriture de flux "<<".
Compilateur     : Mingw-w64 g++ 8.1.0
-------------------------------------------------------------------------------
*/

#ifndef LABO1_MATRIX_HPP
#define LABO1_MATRIX_HPP

#include <iostream>
#include <cstdlib>
#include <ctime>
#include "operation.hpp"

class Matrix;

std::ostream &operator<<(std::ostream &os, const Matrix &matrix);

class Matrix {
public:
    /**
     * Constructeur de la classe matrix.
     * Une exception (runtime_error) est lancée si le modulo vaut 0
     * @param row taille des lignes
     * @param col taille des colonnes
     * @param mod modules
     */
    Matrix(size_t row, size_t col, unsigned mod);

    /**
     * Constructeur de copie
     * @param other matrice à copier
     */
    Matrix(const Matrix &other);

    /**
     * Destructeur de la classe Matrix
     */
    ~Matrix();

    /**
     * Surcharge de l'opérateur d'affectation
     * @param other Matrice à copier
     * @return la matrice après l'affectation
     */
    Matrix &operator=(const Matrix &other);

    /**
     * Surcharge de l'opérateur de flux afin de pouvoir afficher une matrice
     * @param os opérateur de flux
     * @param matrix matrice à afficher
     * @return opérateur de flux utilisé
     */
    friend std::ostream &operator<<(std::ostream &os, const Matrix &matrix);


```

```
277       /**
278        * Addition entre deux matrices, modifie la matrice depuis laquelle la fonction
279        * est appelée.
280        * Cette méthode peut throw une exception invalid_argument.
281        * @param matrix matrice à additionner
282        * @return une référance sur la matrice modifiée après l'addition
283        */
284       Matrix &addItself(const Matrix &matrix);
285
286       /**
287        * Addition entre deux matrices, crée une nouvelle matrice résultant du calcul.
288        * Cette méthode peut throw une exception invalid_argument.
289        * @param matrix matrice à additionner
290        * @return la matrice modifiée après l'addition
291        */
292       Matrix addStaticNew(const Matrix &matrix) const;
293
294       /**
295        * Addition entre deux matrices, crée une nouvelle matrice résultant du calcul.
296        * Cette méthode peut throw une exception invalid_argument.
297        * @param matrix matrice à additionner
298        * @return un pointeur sur la matrice modifiée après l'addition
299        */
300       Matrix *addDynamicNew(const Matrix &matrix) const;
301
302       /**
303        * Soustraction entre deux matrices, modifie la matrice depuis laquelle la
304        * fonction est appelée.
305        * Cette méthode peut throw une exception invalid_argument.
306        * @param matrix matrice à soustraire
307        * @return une référance sur la matrice modifiée après la soustraction
308        */
309       Matrix &subItself(const Matrix &matrix);
310
311       /**
312        * Soustraction entre deux matrices, crée une nouvelle matrice résultant du
313        * calcul.
314        * Cette méthode peut throw une exception invalid_argument.
315        * @param matrix matrice à soustraire
316        * @return la matrice modifiée après la soustraction
317        */
318       Matrix subStaticNew(const Matrix &matrix) const;
319
320       /**
321        * Soustraction entre deux matrices, crée une nouvelle matrice résultant du
322        * calcul.
323        * Cette méthode peut throw une exception invalid_argument.
324        * @param matrix matrice à soustraire
325        * @return un pointeur sur la matrice modifiée après la soustraction
326        */
327       Matrix *subDynamicNew(const Matrix &matrix) const;
328
329       /**
330        * Multiplication entre deux matrices, modifie la matrice depuis laquelle la
331        * fonction est appelée.
332        * Cette méthode peut throw une exception invalid_argument.
333        * @param matrix matrice à soustraire
334        * @return une référance sur la matrice modifiée après la soustraction
335        */
336       Matrix &multItself(const Matrix &matrix);
337
338       /**
339        * Multiplication entre deux matrices, crée une nouvelle matrice résultant du
340        * calcul.
341        * Cette méthode peut throw une exception invalid_argument.
342        * @param matrix matrice à multiplier
343        * @return la matrice modifiée après la multiplication
344        */
345       Matrix multStaticNew(const Matrix &matrix) const;
```

```
346
347         /**
348          * Multiplication entre deux matrices, crée une nouvelle matrice résultant du
349          * calcul.
350          * Cette méthode peut throw une exception invalid_argument.
351          * @param matrix matrice à multiplier
352          * @return un pointeur sur la matrice modifiée après la multiplication
353          */
354         Matrix *multDynamicNew(const Matrix &matrix) const;
355
356     private:
357         /**
358          * Génère une matrice possédant des valeurs aléatoires
359          */
360         void generateMatrix();
361
362
363         /**
364          * Retourne un élément de la matrice, renvoi 0 si les index sont en dehors de
365          * la matrice
366          * @param row le numéro de la ligne
367          * @param col le numéro de la colonne
368          * @return l'élément se trouvant aux index passés en paramètre
369          */
370         unsigned getVal(size_t row, size_t col) const;
371
372         /**
373          * Change la taille de la matrice
374          * @param row nouveau nombre de ligne
375          * @param col nouveau nombre de colonne
376          */
377         void changeSizeValues(size_t row, size_t col);
378
379         /**
380          * Désalloue et détruit le tableau de tableau possédant les valeurs de la matrice
381          */
382         void deleteValues();
383
384         /**
385          * Applique une opération passée en paramètre à tous les éléments de deux
386          * matrices.
387          * @param matrix
388          * @param op
389          * @return
390          */
391         Matrix *applyOperator(const Matrix &matrix, Operation *op);
392
393         /**
394           * Remplace les valuers de la matrice actuelle par les valeurs de la matrice
395           * passée en paramètre.
396           * @param other
397           */
398         void replaceValues(const Matrix &other);
399
400         size_t row, col;
401         unsigned int mod;
402         unsigned **values;
403     };
404
405     #endif //LABO1_MATRIX_HPP
406
407
408
409
410
411
412
413
414
```

```cpp
/*
-------------------------------------------------------------------------------
Nom du fichier  : matrix.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant l'implémentation de la classe Matrix.
Compilateur     : Mingw-w64 g++ 8.1.0
-------------------------------------------------------------------------------
*/

#include "matrix.hpp"
#include "add.hpp"
#include "substract.hpp"
#include "multiply.hpp"
#include "utils.hpp"

Matrix::Matrix(size_t row, size_t col, unsigned mod) : row(row), col(col), mod(mod){
    if(mod == 0){
        throw std::runtime_error("Modulo invalide");
    }
    generateMatrix();
}

Matrix::Matrix(const Matrix &other){
    row = other.row;
    col = other.col;
    mod = other.mod;
    values = new unsigned*[row];
    for (size_t i = 0; i < row; ++i) {
        values[i] = new unsigned[col];
    }
    replaceValues(other);
}

Matrix::~Matrix() {
    deleteValues();
}



void Matrix::deleteValues() {
    for (size_t i = 0; i < row; ++i) {
        delete[] values[i];
    }
    delete[] values;
}

void Matrix::replaceValues(const Matrix &other){
    for (size_t i = 0; i < row; ++i) {
        for (size_t j = 0; j < col; ++j) {
            values[i][j] = other.getVal(i,j);
        }
    }
}

Matrix &Matrix::operator=(const Matrix &other) {
    if(this != &other){

        changeSizeValues(other.row,other.col);
        this->row = other.row;
        this->col = other.col;
        this->mod = other.mod;
        replaceValues(other);
    }
    return *this;
}
```

```
484 std::ostream &operator<<(std::ostream &os, const Matrix &matrix){
485     for (size_t i = 0; i < matrix.row; ++i) {
486         for (size_t j = 0; j < matrix.col; ++j) {
487             os << matrix.getVal(i,j) << " ";
488         }
489         os << std::endl;
490     }
491     return os;
492 }
493
494 void Matrix::generateMatrix() {
495     values = new unsigned*[row];
496     for (size_t i = 0; i < row; ++i) {
497         values[i] = new unsigned[col];
498         for (size_t j = 0; j < col; ++j) {
499             values[i][j] = Utils::randomNumber(mod);
500         }
501     }
502 }
503
504 Matrix& Matrix::addItself(const Matrix &matrix){
505     static Add* add = new Add();
506     applyOperator(matrix,add);
507     return *this;
508 }
509
510 Matrix Matrix::addStaticNew(const Matrix &matrix) const{
511     Matrix m(*this);
512      m.addItself(matrix);
513     return m;
514 }
515
516 Matrix *Matrix::addDynamicNew(const Matrix &matrix) const{
517     Matrix* m = new Matrix(*this);
518     try{
519         m->addItself(matrix);
520     } catch(const std::invalid_argument& e){
521         delete m;
522         throw;
523     }
524     return m;
525 }
526
527 Matrix& Matrix::subItself(const Matrix &matrix) {
528     static Substract* sub = new Substract();
529     applyOperator(matrix,sub);
530     return *this;
531 }
532 Matrix Matrix::subStaticNew(const Matrix &matrix) const{
533     Matrix m(*this);
534      m.subItself(matrix);
535     return m;
536 }
537
538 Matrix *Matrix::subDynamicNew(const Matrix &matrix) const{
539     Matrix* m = new Matrix(*this);
540     try{
541         m->subItself(matrix);
542     } catch(const std::invalid_argument& e){
543         delete m;
544         throw;
545     }
546     return m;
547 }
548 Matrix& Matrix::multItself(const Matrix &matrix){
549     static Multiply* mult = new Multiply();
550     applyOperator(matrix,mult);
551     return *this;
552 }
```

```cpp
Matrix Matrix::multStaticNew(const Matrix &matrix) const{
    Matrix m(*this);
     m.multItself(matrix);
    return m;
}

Matrix *Matrix::multDynamicNew(const Matrix &matrix) const{
    Matrix* m = new Matrix(*this);
    try{
        m->multItself(matrix);
    } catch(const std::invalid_argument& e){
        delete m;
        throw;
    }
    return m;
}

void Matrix::changeSizeValues(size_t row, size_t col){
    if(this->row == row && this->col == col) {
        return;
    }
        unsigned** newValues = new unsigned*[row];
    for (size_t i = 0; i < row; ++i) {
        newValues[i] = new unsigned[col];
    }
    for (size_t i = 0; i < row; ++i) {
        for (size_t j = 0; j < col; ++j) {
            newValues[i][j] = this->getVal(i,j);
        }
    }
    deleteValues();
    this->row = row;
    this->col = col;
    values = newValues;
}

Matrix* Matrix::applyOperator(const Matrix &matrix, Operation* op){
    if(this->mod != matrix.mod){
        throw std::invalid_argument("Les modulos des deux matrices ne sont pas "
                                    "egaux");
    }
    changeSizeValues(std::max(this->row, matrix.row),std::max(this->col,matrix.col));

    for (size_t i = 0; i < this->row; ++i) {
        for (size_t j = 0; j < this->col; ++j) {
            values[i][j] = Utils::floorMod(op->apply(this->getVal(i,j), matrix
            .getVal(i,j)),mod);
        }
    }
    return this;
}

unsigned Matrix::getVal(size_t row, size_t col) const{
    if(row >= this->row || col >= this->col){
        return 0;
    }
    return values[row][col];
}
```

```
/*
-----------------------------------------------------------------------------
Nom du fichier  : operation.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant la déclaration de la classe opération ainsi
                  que de sa méthode "apply" qui permettra d'effectuer une opération
                  sur des matrices. La méthode "apply" devra être redéfinie dans
                  les sous-classes.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/

#ifndef LABO1_OPERATION_HPP
#define LABO1_OPERATION_HPP


class Operation {
public:
    virtual long long apply(unsigned a, unsigned b) = 0;
};


#endif //LABO1_OPERATION_HPP

/*
-----------------------------------------------------------------------------
Nom du fichier  : add.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant la déclaration de la classe add. Cette
                  dernière redéfini la méthode "apply" afin de permettre de faire
                  une addition entre deux matrices.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/

#ifndef LABO1_ADD_HPP
#define LABO1_ADD_HPP

#include "operation.hpp"


class Add : public Operation{
public:
    long long apply(unsigned a, unsigned b);
};


#endif //LABO1_ADD_HPP

/*
-----------------------------------------------------------------------------
Nom du fichier  : add.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant l'implémentation de la classe add.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/

#include "add.hpp"
long long Add::apply(unsigned a, unsigned b){
    return (long long) a + (long long) b;
}
```

```
/*
------------------------------------------------------------------------------
Nom du fichier  : substract.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant la déclaration de la classe substract. Cette
                  dernière redéfini la méthode "apply" afin de permettre de faire
                  une soustraction entre deux matrices.
Compilateur     : Mingw-w64 g++ 8.1.0
------------------------------------------------------------------------------
*/

#ifndef LABO1_SUBSTRACT_HPP
#define LABO1_SUBSTRACT_HPP

#include "operation.hpp"

class Substract : public Operation{
public:
    long long apply(unsigned a, unsigned b);
};


#endif //LABO1_SUBSTRACT_HPP


/*
------------------------------------------------------------------------------
Nom du fichier  : substract.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant l'implémentation de la classe substract.
Compilateur     : Mingw-w64 g++ 8.1.0
------------------------------------------------------------------------------
*/

#include "substract.hpp"
long long Substract::apply(unsigned a, unsigned b){
    return (long long) a - (long long) b;
}

/*
------------------------------------------------------------------------------
Nom du fichier  : multiply.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant la déclaration de la classe multiply. Cette
                  dernière redéfini la méthode "apply" afin de permettre de faire
                  une multiplication composante par composante entre deux matrices.
Compilateur     : Mingw-w64 g++ 8.1.0
------------------------------------------------------------------------------
*/
#ifndef LABO1_MULTIPLY_HPP
#define LABO1_MULTIPLY_HPP

#include "operation.hpp"

class Multiply : public Operation{
public:
    long long apply(unsigned a, unsigned b);
};



#endif //LABO1_MULTIPLY_HPP
```

```cpp
/*
-----------------------------------------------------------------------------
Nom du fichier  : multiply.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 03.03.2022
Description     : Fichier contenant l'implémentation de la classe multiply.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/

#include "multiply.hpp"
long long Multiply::apply(unsigned a, unsigned b){
    return (long long) a * (long long) b;
}

/*
-----------------------------------------------------------------------------
Nom du fichier  : utils.hpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 09.03.2022
Description     : Fichier contenant la déclaration de fonctions utils. Ce dernier
                  défini des fonctions telles que floorMod (inspirée du
                  java) et randomNumber permettant de générer un entier.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------
*/

#ifndef LABO1_UTILS_HPP
#define LABO1_UTILS_HPP

#include <cstdlib>
class Utils {
public:
    /**
     * Génère un entier alétoire modulo la valeur passée en paramètre
     * @param mod est le modulo à appliquer sur la valeur aléatoire.
     * @return une valeure entre [0, mod - 1]
     */
    unsigned static randomNumber(unsigned mod);

    /**
     * Effectue un modulo mathématique entre 2 valeurs passées en paramètres. Le
     * résultat ne peut pas être négatif.
     * @param a
     * @param b
     * @return le modulo des deux paramètres
     */
    unsigned static floorMod(long long a, unsigned b);
};
#endif //LABO1_UTILS_HPP
```

```cpp
/*
-----------------------------------------------------------------------------------
Nom du fichier  : utils.cpp
Auteur(s)       : Alexandre Jaquier, Jonathan Friedli
Date creation   : 09.03.2022
Description     : Fichier contenant l'implémentation de fonctions utilitaires.
Compilateur     : Mingw-w64 g++ 8.1.0
-----------------------------------------------------------------------------------
*/

#include "utils.hpp"

unsigned Utils::randomNumber(unsigned mod) {
    return (unsigned)rand() % mod;
}


unsigned Utils::floorMod(long long a, unsigned b) {
    a %= (long long)b;
    return unsigned (a < 0 ? a + (long long)b : a);
}

```