English version follows French version.

Ce TP comporte trois parties : création d'une librairie implémentant $GF(2,8)$, création d'une librairie pour des polynômes à 1 variable, utilisation des deux pour des codages, pouvant déboucher sur des codes de Reed-Solomon.

1 Implémentation de $GF(2,8)$

Les éléments de ce corps fini seront représentés par un entier sur 8 bits

1. représentation additive : les coefficients d'un polynôme modulo un polynôme irréductible de degré 8 que l'on choisira de plus primitif (les éléments inversibles du corps sont alors les x^0 à x^{254} modulo le polynôme).
2. représentation multiplicative : on indique l'exposant de x ou 255 pour le 0 du corps.

On utilisera un logiciel pour déterminer un polynôme irréductible primitif de degré 8 (par exemple avec la commande `GF` de Xcas). Le polynôme primitif est codé par un entier 9 bits (`unsigned short int`).

On crée une classe avec un membre statique contenant le polynôme primitif, deux tables de conversion de la représentation additive à la représentation multiplicative et réciproquement. La table de conversion de la représentation multiplicative à la représentation additive est créée à l'initialisation de la première instance d'un élément du corps fini de la manière suivante : on multiplie par x la représentation additive de l'élément précédent ce qui revient à effectuer un shift de une position vers la gauche, puis on divise par le polynôme irréductible primitif modulo 2, ce qui revient à partir du passage de x^7 à x^8 à faire un ou exclusif avec le représentant entier 8 bits du polynôme irréductible. On calcule la table inverse en inversant la permutation.

On code alors les `operator +, -, *, /` en passant en représentation multiplicative et inversement pour les deux derniers opérateurs.

On séparera les implémentations dans un fichier source C++, et les déclarations dans un fichier header C++. Puis on génerera une librairie C++ statique (voire dynamique) à l'aide d'un Makefile (outils : `g++ -c, ar, ranlib, make, LD_LIBRARY_PATH` pour les librairies dynamiques) avec un programme de tests utilisant la librairie (outils `gdb`, éventuellement `valgrind` pour les erreurs de mémoire récalcitrantes). Ne pas oublier de commenter le fichier header pour les utilisateurs de la librairie

2 Polynômes

A une variable, représentation dense (liste des coefficients par ordre croissant ou décroissant, en utilisant par exemple le conteneur `vector` de la Standard Template Library). On implémentera les opérations fondementales, soit directement comme polynôme sur la classe créée précédemment (en en faisant une librairie), soit avec des templates et une librairie instanciant le type des coefficients dans le corps fini.

3 Utilisation

A l'aide des libraires précédentes, créer un petit programme de codage avec test de bonne transmission par blocs, avec multiplication par un polynôme de degré 7 et test de la divisibilité à l'arrivée.

Une fois cela terminé, vous pouvez aussi voir
`http://www-fourier.ujf-grenoble.fr/~parisse/crypto/gf.pdf`
et implémenter un code permettant la correction (codes correcteurs de Reed-Solomon).

Correction ou/et aide au démarrage :
`http://www-fourier.ujf-grenoble.fr/~parisse/crypto/f256.tgz`

This text contains 3 parts : creation of a library to handle $GF(2,8)$, creation of a library for polynomials in one variable, combining both of them for writing encoding/decoding application, which might end up implementing Reed-Solomon codes.

4 Implementation of $GF(2,8)$

Each element of the finite field will be represented by a 8 bit integer :

1. in additive representation : each bit represent a coefficient of a polynomial over $\mathbb{Z}/2$ modulo an irreducible polynomial of degree 8. In addition, we will choose the irreducible polynomial to be primitive, hence the invertible elements of the field are x^0 to x^{254} modulo the polynomial. The polynomial will be represented by a 9-bit integer (`unsigned short int`).
2. multiplicative representation : a 8-bit unsigned integer, between 0 and 254 the exponent of x, 255 for the 0 of the finite field.

One can find a primitive irreducible polynomial using a computer algebra system (e.g. using the `GF` command of Xcas).

The field will be represented using a static member for the polynomial, and two table of conversions from/to additive/multiplicative representation. The conversion table from multiplicative to additive representation will be created at the first instantiation of an element of the field, like this : multiply by x the additive representation of the preceding element (this is a left shift of one position), euclidean division by the polynomial modulo 2 (starting at power $x^7 \times x$, this is an exclusive or with the shifted integer). The conversion table from additive to multiplicative representation will be created by inverting the permutation.

Implement `operator +, -, *, /`, using the conversion tables for the two last operations.

Implementation and header declarations shall be separated, the last one being in a `.h` declaration file. Then a static library will be created (or even a dynamic library), using a Makefile (tools `g++ -c, ar, ranlib, make, LD_LIBRARY_PATH`), with a test program using the library (tools `gdb`, or even `valgrind` for hard to debug memory errors). The header file shall be commented for the library users.

5 Polynomials

One dimension polynomials shall be represented by the list of coefficients (in decreasing or increasing order), using an adequate container, e.g. `vector` of the Standard Template Library. Fundamental arithmetic operations shall be implemented either directly as a library using the created class for the field or using templates (the coefficient type being the template parameter) and a library instancing the template with the coefficient type of the preceding section.

6 Using the libraries

Make a small program of transmission by bloc that checks that the received bloc is correct using multiplication by a polynomial of degree 7 (the check being that the remainder of the euclidean division by this polynomial is 0).

Implement Reed-Solomon codes. See (in French)
`http://www-fourier.ujf-grenoble.fr/~parisse/crypto/gf.pdf`

For hints and parts of implementation see :
`http://www-fourier.ujf-grenoble.fr/~parisse/crypto/f256.tgz`