Informatique MP. TP MINITEL

Antoine MOTEAU
antoine.moteau@wanadoo.fr

Dernière rédaction : Aout 2005

. . . /minitel-C.tex (1988 . . . 1996 . . . 2003 . . . 2012 . . .)
. . . /minitel-C.tex Compilé le samedi 10 mars 2018 à 12h 10m 29s avec LaTeX.
Compilable avec LaTeX, PDFLaTeX, LuaLaTeX, XeLaTeX.

Ordre du TP : début d'année (TP 1 ou 2)

Eléments utilisés :

- Représentation binaire des entiers, code ASCII 7bits ou 8 bits
- Quotient de l'anneau $\left(\frac{\mathbb{Z}}{2\mathbb{Z}}[X],+,\times\right)$ par l'idéal engendré par $G = X^7 + X^3 + 1$
- Listes : Polynômes sous forme de liste de coefficients (non creux)
- Compléments éventuels : `big_int`

Documents relatifs au TP :

- Texte du TP : .tex, .dvi, .pdf (Mintel-C pour la version C et Minitel-S pour la version altérée S)
- Squelette de programme Caml : MinitelS.ml
- Programme corrigé Caml : MinitelC.ml
- éventuellement, documentation sur les `big_int`

Distribution pour la réalisation du TP :

- dossier : MINITEL
- contenu :
  - Minitel-S.pdf **un extrait de** Minitel-Cpdf
    - sans la page de garde
    - sans le source du corrigé

    A reconstruire depuis Minitel-C.tex, Minitel-S.dvi à chaque TP.
  - MinitelS.ml

Remarques générales . . .

- On ne traite dans le TP que la partie du Codage CRC et du décodage CRC.
- Pour simplifier, on considère des listes de "bits" (codés `int`) au lieu des `big_int` avec des vrais bits, mais les algorithmes s'adaptent sans trop de difficultés au cas des `big_int` et au travail sur leur représentation binaire physique réelle. Voir document (ancien) avec les big_int

Pour compléter le TP, en fonctionnement exact Minitel, il reste à organiser la transmission générale

- Découpage des pages de caractères en paquets
- Représentation ASCII 8 bits des paquets sous forme de `big_int`
- Codage CRC en utilisant des `big_int`
- Transmission sous forme de flux de `big_int`
- Gestion de la réception du flux . . .
  - Décodage CRC en utilisant des `big_int`
  - Corrections d'erreurs
- Regroupement en pages . . .

Remarque. Pour arriver à des algorithmes vraiment efficaces sur les `big_int`, il faut connaître la structure exacte (ou du moins en avoir la sensation) de l'implantation `CaML` des `big_int` .

Antoine MOTEAU – Lycée POTHIER – Option Informatique MP
TP n^o

CODES CORRECTEURS : CODAGE PAR REDONDANCE CYCLIQUE MINITEL

Introduction

Lors de la transmission de données sur un canal de communication (cable à paire de cuivre, wifi, réseau hertzien, fibre optique, ...), même avec une configuration matérielle correcte, il est inévitable que des erreurs se produisent. Les erreurs, rares mais toujours présentes, peuvent entraîner de graves dysfonctionnements.

La communication avec certaines imprimantes en donne parfois (souvent ?) un exemple : un seul mot de commande erroné provoque un changement de mode d'impression et éventuellement l'édition d'une quantité de pages couvertes de signes cabalistiques à la place du document attendu. Il est alors conseillé de vérifier la configuration matérielle : cable défectueux, trop long, cable non blindé passant à proximité d'une source perturbante, perturbation radioélectrique, ... Mais des erreurs peuvent se produire aussi dans des conditions quasi-"normales".

Pour pouvoir communiquer efficacement, il faut être capable

1. au minimum, de détecter les erreurs.
   La détection se faisant à l'aide d'informations de contrôle ajoutées aux données et la correction étant réalisée par retransmission de tout ou partie des données, ce qui diminue le débit ;
2. ou mieux, d'auto-corriger certaines erreurs.
   La correction se faisant à l'aide d'informations de correction ajoutées aux données.

   Avec auto-correction, on peut passer d'une erreur (en moyenne) sur une page de 1000 caractères à seulement une erreur non corrigée sur plusieurs milliers de pages ... .

Communications internet utilisant tout ou partie des cablages cuivre existants :

- L'Asymmetric Digital Subscriber Line (ADSL), "liaison numérique asymétrique", est une technique de communication qui permet d'utiliser une ligne téléphonique ou une ligne RNIS pour transmettre et recevoir des données numériques de manière indépendante du service téléphonique proprement dit (contrairement aux modems dits analogiques). Cette technologie est mise en œuvre par les fournisseurs d'accès à Internet pour le support des accès "haut-débit".

  La technologie ADSL implique un débit asymétrique, c'est-à-dire que le débit est plus important dans un sens de transmission que dans l'autre, au contraire de la technologie SDSL pour laquelle le débit est symétrique. Pour le particulier, cela se traduit pour l'ADSL par un débit montant (upload) plus faible que le débit descendant (download), généralement d'un facteur de l'ordre de 5 à 20.

  En France, le lancement commercial de l'ADSL a été effectué par France Telecom Interactive en 1999.

  Pour augmenter le débit des communications ADSL, on peut cumuler plusieurs stratégies :

  - Amélioration des supports matériels de communication, ce qui est très coûteux. Par exemple, augmentation de la section des cables de cuivre, utilisation (complète ou partielle) de la fibre optique.
  - Amélioration ou remplacement des équipements, ce qui est très coûteux. Dans les locaux de l'autocommutateur public, l'équipement qui traite les signaux ADSL d'un groupe d'abonnés s'appelle un DSLAM (pour Digital Subscriber Line Access Multiplexer). Les DSLAM regroupent plusieurs centaines, voire plusieurs milliers de cartes électroniques de filtrage.
- Amélioration théorique et logicielle du traitement des signaux (informations datant de 2016).
  - Le VDSL (Very-high-bit-rate Digital Subscriber Line) ...
  - Le VDSL2 (Very-high-bit-rate Digital Subscriber Line 2), successeur du VDSL, est un protocole de transmission de données à haut débit vers un abonné à travers une paire de fils de cuivre. Parmi les améliorations notables, la vitesse maximale théorique passe à 100 Mbit/s en full-duplex, et la distance entre l'utilisateur et le DSLAM est portée à 3 500 mètres.

    La technologie VDSL2 peut faire partie d'une offre dite FTTD (Fiber To The Door), fibre optique jusqu'au palier. Un DSLAM raccordé en fibre optique est installé dans les immeubles et les logements sont raccordés en VDSL2 sur le câblage cuivre existant sans intervention chez les abonnés. Du fait des longueurs réduites de la partie terminale, cette technologie permet d'atteindre un débit de 100 Mbps. Ce mode de raccordement est expérimenté en France à partir de 2013. En 2015, le comité d'experts Cuivre se prononce en faveur de ce mode de raccordement en France.
  - La technologie DSM (Dynamic Spectrum Management), essentiellement théorique et logicielle, est un système qui élimine les bruits parasites (crosstalk/diaphonie) sur la ligne.

L'étude des principes de fonctionnement de l'ADLS dépasse largement le cadre d'un TP. On va se limiter à l'étude d'une technologie plus simple mais significative, précursive de l'ADSL : le Minitel (Médium interactif par numérisation d'information téléphonique). .

Le Minitel (disparu le 30 juin 2012).

Le Minitel est un terminal destiné à la connexion au service français de Vidéotex (service baptisé Télétel), utilisant le réseau téléphonique commuté (liaison par fils de cuivre). Il a été développé par le Ministère des Postes et Télécommunications et utilisé en France, essentiellement dans les années 1980 et 1990, avant d'être supplanté par l'accès à Internet. Par métonymie, le mot Minitel a fini par désigner l'ensemble du service Vidéotex en France ainsi que les éléments de réseau (concentrateurs, points d'accès) destinés à rendre ce service.

La fin du Minitel, plusieurs fois annoncée et toujours repousée, la dernière fois au 30 septembre 2011, a été définitive le 30 juin 2012, bien que 2 millions de personnes l'aient encore utilisé en 2010.

Le Minitel, lancé par France Télécom en 1982, a connu son apogée en 2002, équipant alors neuf millions de foyers et entreprises. Mais il n'a jamais réussi à s'exporter au-delà des frontières françaises.

- Le chiffre d'affaires du Minitel a atteint son point culminant à la fin des années 1990, début 2000, avec un milliard d'euros de revenus annuel, mais il n'a cessé de décliner depuis.
- En février 2009, selon le Groupe France Télécom, le réseau de Minitel enregistre encore 10 millions de connexions mensuelles sur 4 000 codes de services Vidéotex, dont 1 million sur le 3611 (annuaire électronique).
- Fin 2010, il ne restait plus que huit cent dix mille terminaux classiques en circulation. Le service était par ailleurs utilisé par neuf cent cinquante mille personnes sur ordinateur, grâce à un logiciel lancé en 2000 qui permettait d'y accéder via Internet, mais qui va également disparaître.
  Le chiffre d'affaire est tombé en 2010 à 30 millions d'euros bruts.

Dans ce TP, on présente le principe adopté pour le fonctionnement du Minitel, qui constitue une bonne introduction aux principes de télé-communication :

- simple (relativement . . .),
- représentatif des techniques mises en œuvre dans les processus de communication,
- robuste : utilisation de lignes téléphoniques ordinaires, même très anciennes,
- lent (retransmission de pages complètes en cas d'erreurs non corrigées) ;

et qui utilise comme supports théoriques

- l'arithmétique binaire, à partir de la représentation binaire des entiers :
  un entier $u = b_i \times 2^i + b_{i-1} \times 2^{i-1} + \cdots + b_j \times 2^j + \cdots + b_1 \times 2^1 + b_0 \times 2^0$ $\begin{cases} \text{les coefficients } b_j \text{ (poids ou } \underline{\textbf{bits}}\text{)} \\ \text{valant 0 ou 1} \end{cases}$
  est représenté par la <u>séquence</u> $b_i\, b_{i-1} \cdots b_1\, b_0$, avec les **<u>poids forts</u>** en tête ("à gauche")
  Remarques.
  - lorsque la longueur i de la représentation est imposée, cela oblige à considérer des poids forts nuls
  - avec une longueur imposée $i = 8$, on parlera d'**<u>octets</u>** .
- la théorie des groupes, anneaux, corps finis, espaces vectoriels, polynômes en arithmétique binaire (éléments issus de la théorie de **Galois**).
- la théorie des probabilités et des statistiques.

On se limite ici au codage et décodage (avec rectification d'erreur) des paquets élémentaires d'information, et on terminera avec des tests de simulation pour la correction d'erreurs de transmission.

Les ressources du TP sont initialement dans le dossier MPx/. . . /TP-Caml/Minitel/,
à <u>copier</u> dans votre dossier personnel. Ensuite, vous trouverez normalement dans <u>votre</u> dossier TP-Caml/Minitel/

- le texte de ce TP au format .pdf : Minitel-S.pdf,
- un squelette de programme Caml (fonctions à compléter et exemples) : MinitelS.ml

1 Fonctionnement du Minitel

1.1 Emission

On envoie des pages qui sont composées de ℓ paquets de 15 caractères $c_1, c_2, \cdots, c_{15}$ du code ASCII court (128 caractères, codés sur 7 bits).

Chaque paquet de 15 caractères, $c_1, c_2, \cdots, c_{15}$, passe à travers plusieurs niveaux de codage :

Niveau 0 : Codage structurel des caractères sous forme binaire ASCII sur 7 bits

$$\begin{aligned}\text{Message}(0) &= c_1, c_2, \cdots, c_{15} \qquad : 15 \times (7 \text{ bits}) \\ &= \left(b_6 \cdots b_0\right), \left(b_{2\times 8-2} \cdots b_{2\times 8-8}\right), \cdots, \left(b_{15\times 8-2} \cdots b_{15\times 8-8}\right) \\ &= \text{Information}(1), \text{ d'entrée dans le niveau 1}\end{aligned}$$

Niveau 1 : Contrôle de parité local pour chaque caractère :

A chaque caractère c_i (7 bits), $c_i = b_{i\times 8-2} \cdots b_{i\times 8-8}$, on ajoute un bit de de contrôle de parité, $b_{i\times 8-1}$ en poids fort, pour obtenir un octet C_i dont la somme des 8 bits est un multiple de 2.

Si on change l'ordre d'écriture (ce n'est pas obligatoire, mais c'est confortable), on a une correspondance numérique binaire à indices contigus :

$$\begin{aligned}\text{Message}(1) &= C_{15}, C_{14}, \cdots, C_2, C_1 \qquad : 15 \times (\text{ octet}) \text{ juxtaposés } = 120 \text{ bits} \\ &= \left(b_{15\times 8-1} \cdots b_{15\times 8-8}\right), \left(b_{14\times 8-1} \cdots b_{14\times 8-8}\right), \cdots, \left(b_7 \cdots b_0\right) \\ &= \text{Information}(2), \text{ d'entrée dans le niveau 2}\end{aligned}$$

Niveau 2 : **Codage avec contrôle de redondance cyclique (CRC)** (détaillé plus loin) :

Ce codage, appliqué au résultat précédent (120 bits), produit une nouvelle séquence, de $120+7 = 127$ bits :

$$\begin{aligned}\text{Message}(2) &= 7 \text{ bits } + 15 \times \text{ octets } = 127 \text{ bits} \\ &= \left(a_{16\times 8-2} \cdots a_{16\times 8-8}\right), \left(a_{15\times 8-1} \cdots a_{15\times 8-8}\right), \left(a_{14\times 8-1} \cdots a_{14\times 8-8}\right), \cdots, \left(a_7 \cdots a_0\right) \\ &= \text{Information}(3), \text{ d'entrée dans le niveau 3}\end{aligned}$$

Remarque. les 7 bits de poids fort constituent les 7 bits de poids faible d'un 16-ième octet.

Niveau 3 : Contrôle de parité global :

On introduit un bit de contrôle de parité global, 8-ième bit en poids fort du 16-ième octet, tel que la somme des bits de tous les octets soit multiple de 2.

$$\begin{aligned}\text{Message}(3) &= (1+7) + 15 \times (8) = 16 \times \text{ (octets) } = 128 \text{ bits} \\ &= \left(a_{16\times 8-1} a_{16\times 8-2} \cdots a_{16\times 8-8}\right), \left(a_{15\times 8-1} \cdots a_{15\times 8-8}\right), \cdots, \left(a_7 \cdots a_0\right) \\ &= \text{Information}(4), \text{ d'entrée dans le niveau 4}\end{aligned}$$

Niveau 4 : Contrôle de fond :

Ajout d'un 17 octet (0000 0000 systématique, par exemple), en poids fort, pour le contrôle de fond :

$$\begin{aligned}\text{Message}(4) &= 1 \text{ (octet) } + 16 \times \text{ (octets) } = 17 \times \text{ (octets) } = 136 \text{ bits} \\ &= \left(a_{17\times 8-1} \cdots a_{17\times 8-8}\right), \left(a_{16\times 8-1} \cdots a_{16\times 8-8}\right), \left(a_{15\times 8-1} \cdots a_{15\times 8-8}\right), \cdots, \left(a_7 \cdots a_0\right) \\ &= a_{17\times 8-1}\, a_{17\times 8-2} \cdots a_{17\times 8-8}\, a_{16\times 8-1} \cdots a_{16\times 8-8}\, a_{15\times 8-1} \cdots\cdots\cdots a_8\, a_7 \cdots a_0\end{aligned}$$

C'est cette dernière séquence qui est envoyée sur le canal de transmission, bit par bit, comme représentant du paquet intitial des 15 caractères $c_1, c_2, \cdots, c_{15}$.

1.2 Réception

Pour chaque paquet $c_1, c_2, \cdots, c_{15}$, la séquence reçue, traitée réciproquement, doit fournir la séquence initiale des 15 caractères $c_1, c_2, \cdots, c_{15}$ du code ASCII court (7 bits).

Sur un canal de transmission, en temps normal, les erreurs étant rares, isolées et aléatoires,

- le contrôle de fond (17^e octet) permet de détecter les erreurs nombreuses (ligne hors service) ;
- le contrôle de parité global permet de détecter un nombre impair d'erreurs ;
- le contrôle de redondance cyclique permet la correction d'une seule erreur et la détection de deux erreurs (trois si on renonce à corriger) ;
- le contrôle de parité local permet, en complément, la détection d'une seule erreur par octet.

1.3 Résumé

Si le nombre (moyen) d'erreurs par paquet est

- important : Rien ne passe (octet 17 de contrôle de fond toujours perturbé), ligne coupée.
- nul : communication rapide.
- au plus un : auto-correction, sans retransmission avec une communication assez rapide
- au plus deux : correction par retransmission de la page entière, d'où une communication lente
- faible mais supérieur à deux : le contrôle de redondance cyclique local permet de déceler la plupart des erreurs et la correction se fera par retransmission d'où une communication lente

Il est rare, mais pas impossible, que plusieurs erreurs (au moins quatre) se compensent de telle sorte que tous les contrôles soient valides.

2 Codage et décodage par contrôle de redondance cyclique (CRC)

On traite ici le codage et le décodage d'un paquet, à partir de l'information obtenue à l'issue du niveau 1, où chaque caractère est déjà accompagné de son bit de contrôle de parité.

2.1 Codage

1. L'information à coder est la séquence des $k = 15 \times 8 = 120$ bits d'information :

$$\left(b_{15\times 8-1}\cdots b_{15\times 8-8}\right),\ \left(b_{14\times 8-1}\cdots b_{14\times 8-8}\right),\ \cdots,\ \left(b_7\cdots b_0\right)$$

interprétée comme un polynôme de degré ("au plus") $k-1=119$, à $k=120$ coefficients dans $\frac{\mathbb{Z}}{2\mathbb{Z}}$:

$$b_{15\times 8-1}\,X^{15\times 8-1}+\cdots+b_{15\times 8-8}\,X^{15\times 8-8}+b_{14\times 8-1}\,X^{14\times 8-1}+\cdots+b_7\,X^7+\cdots+b_1\,X+b_0$$

représenté également comme un entier écrit en binaire :

$$b_{15\times 8-1}\,2^{15\times 8-1}+\cdots+b_{15\times 8-8}\,2^{15\times 8-8}+b_{14\times 8-1}\,2^{14\times 8-1}+\cdots+b_7\,2^7+\cdots+b_1\,2+b_0$$

2. **Ce polynôme est multiplié par le polynôme $G=X^7+X^3+1$, de degré $r=7$** ,

ce qui donne un polynôme de degré ("au plus") $n-1=k+7-1=126$, à $n=127$ coefficients :

$$A=a_{16\times 8-2}\,X^{16\times 8-2}+a_{16\times 8-3}\,X^{16\times 8-3}+\cdots+a_7\,X^7+\cdots+a_1\,X+a_0$$

représenté aussi comme un entier écrit en binaire :

$$a_{16\times 8-2}\,2^{16\times 8-2}+a_{16\times 8-3}\,2^{16\times 8-3}+\cdots+a_7\,2^7+\cdots+a_1\,2+a_0$$

3. On ajoute en tête le monôme $a_{16\times 8-1}\,X^{16\times 8-1}$ où

$$a_{16\times 8-1}=\left(\sum_{i=0}^{16\times 8-2} a_i\right) \mod 2 \qquad \text{(bit de parité global)}$$

4. On ajoute en tête $a_{17\times 8-1}\,X^{17\times 8-1}+\cdots a_{17\times 8-8}\,X^{17\times 8-8}$ où

$$a_{17\times 8-1}=\cdots=a_{17\times 8-8}=0 \qquad \text{, pour représenter le 17-ième octet nul (contrôle de fond)}$$

2.2 Transmission

Le message polynomial codé final, de degré ("au plus") $120+7+1+8-1=135$, à $136=17\times 8$ coefficients,

$$a_{17\times 8-1}\,X^{17\times 8-1}+\cdots+a_1\,X+a_0$$

représenté aussi comme un entier écrit en binaire :

$$a_{17\times 8-1}\,2^{17\times 8-1}+\cdots+a_1\,2+a_0$$

est envoyé, bit par bit (coefficient par coefficient), sur le canal de communication.

2.3 Décodage et correction d'une seule erreur

On suppose qu'il n'y a pas plus d'une erreur dans la transmission.

1. Le message codé reçu, $a'_{17\times 8-1}\,2^{17\times 8-1}+\cdots+a'_1\,2+a'_0$, est privé de sa tête (17-ième octet)
2. On enlève encore le bit de poids fort (bit de parité global), pour obtenir le polynôme
$$A'=a'_{16\times 8-2}\,X^{16\times 8-2}+a'_{16\times 8-3}\,X^{16\times 8-3}+\cdots+a'_7\,X^7+\cdots+a'_1\,X+a'_0$$
3. On divise le polynôme A' par le polynôme $G=X^7+X^3+1$
   - Si le reste est nul, le quotient est supposé contenir l'information valide.
   - Sinon, comme on suppose qu'il n'y a qu'une seule erreur, soit e l'indice du coefficient erroné.
     (a) on cherche l'emplacement de l'erreur dans le message A', sachant que :
     - $A'=A+X^e$
     - A' et X^e ont le même reste dans la division par $G=X^7+X^3+1$
     - Les restes de $1,X,\cdots,X^{126}$ dans la division par $G=X^7+X^3+1$ sont tous distincts

     ce qui permet d'identifier l'indice e (du coefficient erroné) sans ambiguïté ;
     (b) **on corrige** le message reçu en remplaçant A' par $A'+X^e=A$;
     (c) on re-calcule le quotient, ce qui donne le message décodé valide attendu, composé des 16 caractères (chaque caractère sur 8 bits, avec bit de parité en poids fort).

Remarque. Il faut aussi faire intervenir le contrôle de parité global puis les contrôles de parité de chaque octet pour conclure efficacement.

3 Erreurs "normales" de transmission (avec octet 17 nul)

Définition 3.1.

Le **poids d'un message** est le nombre de bits non nuls de ce message.
La **distance entre deux messages** est le nombre de bits de même rang qui différent d'un message à l'autre (poids de la différence).

L'information initiale est composée des 15 octets (caractères avec bit de parité) et les messages codés obtenus

- comportent $n=15\times 8+8-1=127$ bits (CRC compris, bit de parité global et 17-ième octet exclus),
- véhiculent $k=15\times 8=120$ bits d'information.

Il y a donc en tout 2^{127} messages codés possibles dont 2^{120} sont valides, c'est à dire correspondent au codage de 15 caractères du code ASCII 7 bits accompagnés de leur bit de parité.

A la réception, le 17-ième octet étant nul, le bit de parité global sur l'ensemble des 16 autres octets permet de détecter les erreurs simples et plus généralement les erreurs en nombre impair.
La distance minimale entre deux messages (de niveau 3), valides et distincts, est $4=2\times 1+1+1$, donc,

- avec une seule erreur, le message reçu est situé à la distance 1 d'un seul message valide.
  (on peut donc corriger une erreur) ;
- avec deux erreurs, le message reçu peut être équidistant de deux messages valides.
  (on peut donc détecter deux erreurs mais pas les corriger) ;

- avec trois erreurs, le message reçu peut se trouver à une distance 1 d'un message valide.
  (trois erreurs peuvent apparaître comme une seule erreur, ce qui fait que, si on renonce à corriger, on détecte trois erreurs) ;
- avec plus de trois erreurs
  - un nombre impair d'erreur donne un bit de parité global erroné, sans que l'on sache s'il s'agit d'une erreur, de trois erreurs ou plus,
  - un nombre pair d'erreurs pourrait passer inaperçu, mais sera souvent détecté à l'aide du contrôle de parité octet par octet (erreurs rares et isolées).

Pour des pages de 10^5 bits (environ), avec une probabilité d'erreur par bit égale à 0.00001 ,
- sans correction, il y a en moyenne une erreur par page (d'où retransmission de la page entière) ;
- avec correction d'une erreur, il y a en moyenne 0.1 erreur par page, soit 10 pages sans erreurs (retransmission d'environ une page sur 10).

4 Justification (partielle) des choix pour le Minitel

On a choisi de prendre $G = 1 + X^3 + X^7$, $r = 7$, $n = 127$ (raisons de simplicité et efficacité).

4.1 Le corps Q_7

- $\dfrac{\mathbb{Z}}{2\mathbb{Z}}$ est identifié à l'ensemble $\{0,1\}$.
- $\left(\dfrac{\mathbb{Z}}{2\mathbb{Z}},+,\times\right)$ est un corps commutatif **-1 = 1 , 1+1 = 0 , a - b = a + b**
- $\left(\dfrac{\mathbb{Z}}{2\mathbb{Z}}[X],+,\times\right)$ est un anneau.
- Le polynôme $G = X^7 + X^3 + 1$ est **premier** dans $\dfrac{\mathbb{Z}}{2\mathbb{Z}}[X]$, $\begin{cases}\text{pas trop gros}\\ \text{pas trop petit}\end{cases}$ ("idéal")

Soit Q_7 le quotient de l'anneau $\left(\dfrac{\mathbb{Z}}{2\mathbb{Z}}[X],+,\times\right)$ par l'idéal engendré par $G = X^7 + X^3 + 1$

- Q_7 est l'ensemble des restes possibles dans la division des polynômes par G.
- Q_7 est muni des opérations "naturelles" (où $\overline{P}$ désigne la classe d'équivalence de P) :
  - internes "quotient" $+$ et $\times$: $\overline{P}+\overline{Q} = \overline{P+Q}$ et $\overline{P}\times\overline{Q} = \overline{P\times Q}$
  - externe $\cdot$ à opérateur dans $\dfrac{\mathbb{Z}}{2\mathbb{Z}}$: $\lambda\cdot\overline{P} = \overline{\lambda\cdot P}$

ce qui donne les structures suivantes :

- $(Q_7,+,\times)$ est un corps (puisque G est premier)
- $(Q_7,+,\cdot)$ est un espace vectoriel sur $\left(\dfrac{\mathbb{Z}}{2\mathbb{Z}},+,\times\right)$, de base $\left(\overline{1},\overline{X},\overline{X^2},\cdots,\overline{X^6}\right)$

Q_7 possède donc 2^7 éléments et $Q_7^* = Q_7 \setminus \{0\}$ possède $2^7 - 1 = 127$ éléments, avec $2^7 - 1 = 127$ qui est **premier**.

$(Q_7^*,\times)$ est un groupe à $2^7 - 1$ éléments, et comme $2^7 - 1$ est premier, ce groupe est un groupe cyclique, de générateur $\overline{X}$, ce qui prouve que

$$Q_7^* = \left\{ \overline{1} , \overline{X} , \overline{X^2} , \cdots , \overline{X^{126}} \right\}$$

On en déduit que les restes de $1, X, \cdots, X^{126}$ dans la division par $X^7 + X^3 + 1$ sont tous non nuls et distincts.

Ainsi, on pourra corriger une erreur lorsque l'on traite des polynômes de degré "au plus" 126, ayant $n = 127$ coefficients.

4.2 Distance entre deux messages codés valides

La capacité de détection et de correction d'erreurs est fonction de la distance entre deux messages codés valides :

Les messages codés par le codage de contrôle de redondance cyclique sont les polynômes de $\frac{\mathbb{Z}}{2\mathbb{Z}}[X]$, de degré strictement inférieur à 127 (à 127 coefficients), et les messages codés valides sont ceux dont le reste dans la division par G est nul.

- La différence entre deux messages codés valides distincts est un message codé valide non nul.
- Tous les messages codés valides non nuls ont au moins trois coefficients non nuls :
  sinon, on aurait des messages codés valides de la forme X^i ou $X^i + X^j$, avec $i, j \in [0 \cdots 126], i < j$.
  - X^i a un reste non nul, donc X^i n'est pas valide
  - X^i et X^j ont des restes non nuls distincts et leur somme (et différence) aura un reste non nul.

Ainsi, la distance entre deux messages codés valides est supérieure ou égale à $3 = 2 \times 1 + 1$.

Remarque. Comme $n - 7 = 120 = 15 \times 8$, on organise les pages en paquets de 15 caractères, et pour faire un compte "rond" d'octets (128 bits), à partir des 127 bits, on ajoute un bit en poids fort (utilisé comme contrôle de parité globale).
L'introduction du bit de contrôle de parité global augmente la distance d'une unité.

5 Mise en œuvre du codage et du décodage

Les informations à traiter (paquets de 15 octets) peuvent être considérées de façon équivalente comme

- des entiers décomposés sous forme binaire (de longueur fixe)
  (les opérations de codage et décodage faisant appels à des calculs sur les entiers).
- des polynômes à coefficients dans $\{0, 1\}$
  (les opérations de codage et décodage faisant appels à des calculs sur les polynômes).

5.1 Représentation, interprétation de l'information

5.1.1 Représentation comme entiers

Le plus grand entier susceptible d'intervenir dans les calculs précédents, $2^{135} + 2^{134} + \cdots + 2 + 1 = 2^{136} - 1$, **dépasse la capacité de représentation entière** des langages courants où les entiers sont, le plus souvent, codés en binaire sur 4 octets (32 bits, dont un pour le signe), ce qui limite le plus grand entier à $2^{32} - 1$.

Pour contourner cette limite, la représentation binaire (de longueur fixe) d'un entier compris entre 0 et $2^{136} - 1$ peut être réalisée

- à l'aide d'une liste de 136 valeurs (entières) 0 ou 1, **"pseudo-bits"**, ce qui mobilise 136×32 bits (plus les liens de liste) pour représenter en fait 136 bits ! C'est inopérant, mais la mise en œuvre des calculs est simple.

  Si l'objectif n'est que de comprendre le fonctionnement, c'est cette solution que l'on prendra.
- par découpage en tronçons de 32 bits, ce qui demande seulement une liste de 5 entiers pour représenter 5×4 octets $= 160$ bits. C'est la technique utilisée dans le Minitel, mais la mise en oeuvre des calculs est un peu compliquée (accès aux bits, gestion des reports).
- **en utilisant une bibliothèque spécialisée dans la gestion des grands nombres entiers** :
  En `CaML`, une bibliothèque introduit le type `big_int` pour représenter des entiers de taille non limitée
  (en fait, il s'agit de la solution précédente de découpage en tronçons de 32 bits, organisés comme liste d'entiers, avec des ressources de calcul déjà programmées).

5.1.2 Interprétation comme polynômes

Si on considère les informations à traiter comme des polynômes à coefficients dans $\{0, 1\}$, on doit mettre en place, **en arithmétique binaire**,

- le codage : multiplication par $1 + X^3 + X^7$ dans $\frac{\mathbb{Z}}{2\mathbb{Z}}[X]$
- le décodage : quotient et reste dans la division par $1 + X^3 + X^7$ dans $\frac{\mathbb{Z}}{2\mathbb{Z}}[X]$

A la place d'algorithmes généraux de multiplication et de division de polynômes (trop coûteux en temps), on utilise des algorithmes qui **exploitent les particularités** du polynôme $1+X^3+X^7$, pour obtenir un temps de calcul le plus faible possible. Ces algorithmes sont adaptés à l'une des représentations choisie :

- liste de (au plus) 136 coefficients entiers (à valeur 0 ou 1), **"pseudo-bits"**, ce qui n'est pas réellement efficace puisque chaque coefficient mobilise 32 bits pour représenter un seul bit ;
- entier long (liste de 5 entiers) ;
- `big_int` .

Bien que le meilleur des choix serait celui de la représentation par entiers découpés en tronçons (liste de 5 entiers ou `big_int`), on choisit ici (c'est plus simple) la représentation à l'aide de listes de (au plus) 136 coefficients entiers (à valeur 0 ou 1), **"pseudo-bits"**, interprétées comme listes de coefficients de polynômes.

Une fois le principe établi, il ne resterait plus qu'à réaliser l'adaptation à la représentation optimale par une liste de 5 entiers.

On ne réalisera ici que les étapes correspondant au codage par CRC et au décodage (avec détection et correction d'erreur), ainsi que la simulation (introduction et correction d'erreurs).

En complément, on peut introduire les étapes conduisant d'un message alphanumérique de 15 caractères (du code ASCII court) au polynôme à coder puis au polynôme codé, ainsi que celles conduisant du polynôme codé reçu au polynôme décodé corrigé puis au message alphanumérique de 15 caractères correspondant ....

5.2 Éléments pour l'interprétation comme polynômes

Un polynôme étant représenté par la liste de ses coefficients, dans l'ordre des puissances croissantes,

- le multiplier par X^m consiste à introduire m zéros en tête de liste.
- le diviser par X^m consiste à
  - prendre la liste des m premiers éléments comme reste
  - la liste des éléments suivants comme quotient

On conviendra

- que le polynôme nul peut être représenté par la liste vide (mais aussi par une liste de zéros).
- que les coefficients dominants d'un polynôme peuvent être nuls. Par exemple :
  [] "=" $[0;0;0;0;0]$ (polynôme 0) et $[1;0;1;1]$ "=" $[1;0;1;1;0;0;0;0;0;0]$ (polynôme $1+X^2+X^3$).

- On ne mettra pas en œuvre ici une politique de gestion de polynômes creux ...
- Bien que les coefficients soient binaires ($\{0,1\}$), **pour simplifier** (illustration de la théorie), on les représentera à l'aide d'entiers, **"pseudo-bits"** (utilisation de 32 bits pour représenter un seul bit !).

Visualisation, conversions :

1. Conversion d'un polynôme (liste de coefficients entiers quelconques, pas seulement 0 ou 1) en une chaîne de caractères représentant l'écriture mathématique usuelle d'un polynôme de la variable X.

```
let string_of_int_poly f =
  let u = aux 0 f
  where rec aux k = function
        []   -> ""                          (* je dis chaîne vide *)
      | a::q -> let p = aux (k+1) q
                in if a = 0 then p
                   else let h = if k = 0 then string_of_int a
                                else   (if a = 1 then ""
                                        else if a = -1 then "-" else string_of_int a )
                                     ^ (if k = 1 then "X" else " X^" ^ (string_of_int k))
                        in if string_length p = 0 then h
                           else if p.[0] = '-' then h ^ p else h ^ "+" ^ p
  in if u = "" then "0" else u
;;
(* string_of_int_poly : int list -> string = <fun> *)

let G = [1; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0];;
print_string (string_of_int_poly G);;   (* -->  1 + X^3 + X^7 *)
```

2. Transformation d'un polynôme (liste de exactement $N \times 8$ "pseudo-bits") en une chaîne de N caractères.

   Remarque. Pour avoir une liste de (exactement) $N \times 8$ "pseudo-bits", on complétera préalablement la queue de liste par des 0 en quantité suffisante.

```
(* Transformation d'une liste de 8 (=1+7) "pseudo-bits" en char (ASCII 7 bits) *)
let char_of_bin_list l = char_of_int ((aux 7 l) land 127)
  where rec aux k = function
     []    -> 0
    | a::q -> (a lsl k) + aux (k-1) q
;;
(* char_of_bin_list : int list -> char = <fun> *)

(* Transformation d'une liste de N * 8 "pseudo-bits" en string de longueur N *)
let rec string_of_bin_list = function
    [] -> ""
  | a0::a1::a2::a3::a4::a5::a6::a7::q -> let lc = [a0;a1;a2;a3;a4;a5;a6;a7] in
    (string_of_bin_list q) ^ (string_of_char (char_of_bin_list lc) )
;;
(* string_of_bin_int_list : int list -> string = <fun> *)
```

3. Transformation d'une chaîne (de N caractères) en polynôme (liste de $N \times 8$ "pseudo-bits") :

   Voir les fonctions :

   - Transformation d'un char ASCII 7, éditable, en "pseudo-octet" (liste de 8 "pseudo-bits"), avec 8^o bit de parité :
     `bin_list_of_char : char -> int list = <fun>`
   - Transformation d'une chaîne de caractères en liste de $N \times 8$ "pseudo-bits" :
     `bin_list_of_string : string -> int list`

   qui sont pré-écrites dans le squelette de programme (en annexe).

Ces fonctions sont à utiliser pour les tests et on pourra également s'en inspirer pour la gestion de listes.

5.3 Codage

Multiplication par $G = X^7 + X^3 + 1$, en arithmétique binaire.

On suppose que les bits d'information (les coefficients du polynôme à multiplier) arrivent logiquement, un à un, **poids forts en tête** (coefficient dominant en tête) donc, pour multiplier par $G = X^7 + X^3 + 1$, on reporte en partie les coefficients arrivant sur ceux qui sont déjà arrivée et traités.

Un polynôme étant représenté par la liste de ses coefficients, dans l'ordre des puissances croissantes, on traitera la queue de liste en premier, avec report partiel du coefficient de tête (poids faible) sur des coefficients de queue déjà traités (poids forts).

Pour multiplier par $X^7 + X^3 + 1$, à l'arrivée de chaque coefficient (nouveau coefficient de poids faible),

- introduire ce coefficient en tête d'une liste correctement initialisée
  (multiplier ce coefficient par 1 et "pousser" d'un cran les anciens déjà multipliés),
- additionner (**modulo 2**) ce coefficient en position X^3 (multiplication par X^3),
- additionner (**modulo 2**) ce coefficient en position X^7 (multiplication par X^7).

Cela se réalisera par la fonction :

```
let rec multiply_byG = function
 | []   -> []
 | a::q -> match multiply_byG q with
             | ....                              (* une ligne à écrire *)
             | ....                              (* une ligne à écrire *)
;;
(* multiply_byG : int list -> int list = <fun> *)

print_string (string_of_int_poly (multiply_byG G) );;
```

5.4 Transmission avec perturbation par une seule erreur (simulation)

Pour simuler une erreur en position e, on transforme le polynôme codé (`int_list`) en ajoutant 1 (**modulo 2**) au coefficient de X^e, l'indice e étant une valeur entière que l'on choisit (éventuellement au hasard) entre 0 et 126 (127 (135)), extrémités comprises.

Remarque. Dans la réalité, le polynôme est représenté par une liste de longueur effectivement 127 (128 (136)), avec des 0 au delà du degré effectif.

Ici, pour des raisons de simplicité de saisie, on ne représente pas toujours les zéros qui sont au delà du degré, **ce qui oblige à générer les 0 qui manquent** lorsque l'erreur doit porter sur une position inexistante.

On écrit donc une fonction qui crée une erreur en position e, en générant les 0 qui manquent :

```
let rec perturbe e = function
  | []   -> ....                          (* une ligne à écrire *)
  | a::q -> ....                          (* une ligne à écrire *)
;;
(* perturbe : int -> int list -> int list = <fun> *)

perturbe 2 G;;       (* test simple *)
```

5.5 Décodage

Division par $G = X^7 + X^3 + 1$, en arithmétique binaire.

On suppose que les bits d'information (les coefficients du polynôme à diviser) arrivent logiquement, un à un, **poids faibles en tête** (coefficient faible en tête) donc, pour diviser par $G = X^7 + X^3 + 1$, on reporte en partie les coefficients arrivant sur ceux qui sont déjà arrivés et traités.

Un polynôme étant représenté par la liste de ses coefficients, dans l'ordre des puissances croissantes, on traitera la queue de liste en premier, avec report de bits de poids forts vers des bits de poids faibles.

Pour diviser par $X^7 + X^3 + 1$, pour chaque coefficient de tête (poids faible !),

- introduire le coefficient en tête d'une liste correctement initialisée (pousser les anciens),
- réduire le terme en X^7 par division :
  - "enlever" (**ajouter modulo 2**) le coefficient de position X^7 au coefficient de position X^3,
  - "enlever" (**ajouter modulo 2**) le coefficient de position X^7 au coefficient de position X^0 ;

  puisque $X^7 = (X^7 + X^3 + 1) \times 1 + X^3 + 1 \mod G$.

Ce qui donne une liste dont on extrait le quotient et le reste :

- la liste des 7 premiers éléments constitue le reste (un reste nul s'exprime par la liste $[0;0;0;0;0;0;0]$),
- la liste des éléments suivants constitue le quotient.

Cela se réalisera par la fonction :

```
let QRdivide_byG f =        (*  f --> quotient, reste *)
  let rq = aux f
       where rec aux = function              (* traitement des bits un à un *)
         | []    -> []
          | a::q -> match ...  with
                      | ... -> ...
                      | ... -> ...
  in match rq with                           (* séparation en quotient, reste *)
        | ... -> ...
        | ... -> ...
;;
(* QRdivide_byG : int list -> int list * int list = <fun> *)      (* 6 lignes à compléter *)

let H = [1;1;0;1;0;1;1;1;0;0;1;0;0;0;1];;     (* conçu comme H = Q G + R *)
print_string (string_of_int_poly H);;
let q, r = QRdivide_byG H;;
print_string (string_of_int_poly q);;
print_string (string_of_int_poly r);;
```

On suppose qu'il y a au plus une seule erreur, détectée par un reste non nul.

- Si le reste est nul, il n'y a pas d'erreur, et le message est le quotient du polynôme reçu.
- Si le reste est non nul, alors on corrige l'erreur :

1. **Recherche de l'indice e du bit erroné :**

   Il suffit de comparer le reste obtenu aux restes successifs de $1, X, X^2, \cdots, X^m, \cdots, X^{126}$ dans la division par X^7+X^3+1, en sachant qu'il y a une (et une seule) erreur hors du bit de parité globale (la parité globale est incorrecte et le reste non nul). Lorsque le reste sera égal au reste de X^m, on aura $e=m$.

   Le reste de $1=X^0$ étant $[1;0;0;0;0;0;0]$, le reste de X^{m+1} s'obtient

   - en "multipliant" le reste de X^m par X (ce qui correspond à un "décalage")
   - puis en réduisant le terme en X^7 :
     - ajout (**modulo 2**) du coefficient de X^7 au coefficient de X^0 et au coefficient de X^3
     - "suppression" du terme en X^7 (on ne s'intéresse pas au quotient)

   puisque $\boxed{X^7 = 1+X^3 \mod G}$.

   Cela se réalisera par la fonction :

```
let bad_indice r = aux 0 [1;0;0;0;0;0;0]   (* Ne termine qu'en cas d'indice erroné *)
  where rec aux m ( ... as rxm) =
     if rxm = r then ...           (* égalité générique sur liste de même longueur *)
     else ...
;;
(* Warning: this matching is not exhaustive
   bad_indice : int list -> int = <fun> *)                (* 3 lignes à compléter *)

let H = [0;0;0;0;0;0;0;0;0;0;1];;  (* X^10 *)
let q, r = QRdivide_byG H;;
bad_indice r;;
```

2. **Correction de la valeur du bit erroné en position e :**

   Il suffit d'appliquer à nouveau la procédure utilisée pour créer une erreur en position e (ce qui rectifie l'erreur) sur le polynôme reçu.

3. **Message reçu corrigé :**

   On ne retient que le quotient du polynôme reçu corrigé, obtenu par la procédure de calcul de quotient et reste (déjà écrite).

5.6 Outils de test : comparaison (artificielle) de la réception avec l'émission

Remarque. L'égalité = de CaML (version $\geqslant 0.74$) est une égalité générique ...

Soient P le polynôme à coder initial et Q le polynôme décodé (corrigé) reçu.

Si le polynôme P a été représenté par une liste de longueur strictement inférieure à $k=120$, l'algorithme d'introduction d'une erreur aléatoire aurait pu allonger la liste transmise par les 0 qui manquent, et la liste du polynôme Q pourrait être plus longue que celle de P ... aie aie aie !

- si P et Q sont représentés par des listes de même longueur, le test pourra se faire par `P = Q ;;`
- sinon, il faut ramener chaque liste à sa longueur minimale, en supprimant les zéros superflus (qui sont en fond de liste ... puisque les poids faibles sont en tête), avant de faire la comparaison générique.

Pour simplifier, dans les tests, on utilisera systématiquement, pour les polynômes à coder ou décodés (donc hors les 8 bits de l'octet de fond, le bit de parité globale, et après réduction des bits de CRC), des listes "polynômes" de longueur exactement $120 = 15 \times 8$ (avec bit de parité compris pour chaque caractère).

6 TESTS

On n'introduit pas le bit de parité globale ni le 17-ième octet de contrôle de fond :

on se limite ici au test du principe de réparation d'une seule erreur.

Un polynôme P de degré au plus $u \leqslant 119$, à coefficients 0 ou 1, sera représenté par une liste d'entiers (0 ou 1) de longueur $v = 120$ exactement.

1. Génération (aléatoire) d'un polynôme à coder, de degré au plus u, à $v > u$ coefficients.

```
let rec genere = function                          (* Non sécurisée (pour tests) *)
       -1, v -> list_of_vect (make_vect v 0)
     | u , v -> (random__int 2):: (genere (u-1, v-1))
;;
(* genere : int * int -> int list = <fun> *)

let H = genere (12,20);;              (* degré au plus 12, liste de longueur 20 *)
print_string (string_of_int_poly H);;
```

2. Enchaînements de test : Codage - Transmission - Décodage correction

```
let P = genere (20, 120);;               (* degré au plus 20, liste de longueur 120 *)
print_string (string_of_int_poly P);;

let A = multiply_byG P;;                  (* codage *)
print_string (string_of_int_poly A);;

let A = perturbe 91 A;;                   (* erreur de transmission sur le bit 91 *)
print_string (string_of_int_poly A);;

let Q, R = QRdivide_byG A;;               (* décodage et détection d'erreur *)
print_string (string_of_int_poly Q);;
print_string (string_of_int_poly R);;

let e = bad_indice R;;                    (* identification de la position d'erreur *)
let A = perturbe e A;;                    (* correction *)

let Q, R = QRdivide_byG A;;               (* décodage et détection d'erreur *)
print_string (string_of_int_poly Q);;
print_string (string_of_int_poly R);;

P = Q;;                                   (* vérification (même longueur 120) *)
```

3. Décodage correction (bit de parité globale et 17-ième octet déjà exclus)

Identifier le message (de 15 caractères) contenu dans le polynôme codé reçu :

$$\begin{aligned} A =& X + X^2 + X^7 + X^{10} + X^{11} + X^{15} + X^{19} + X^{28} + X^{29} + X^{41} + X^{42} + X^{43} + X^{44} + X^{54} + X^{55} \\ &+ X^{56} + X^{57} + X^{60} + X^{61} + X^{62} + X^{63} + X^{67} + X^{68} + X^{70} + X^{73} + X^{74} + X^{76} + X^{77} + X^{78} \\ &+ X^{79} + X^{80} + X^{83} + X^{84} + X^{85} + X^{90} + X^{91} + X^{92} + X^{94} + X^{96} + X^{101} + X^{104} + X^{105} \\ &+ X^{106} + X^{108} + X^{109} + X^{110} + X^{111} + X^{112} + X^{114} + X^{117} + X^{119} + X^{122} + X^{124} \end{aligned}$$

$$\begin{aligned} A =& 01100001001100010001000000001100000000000011111000000000111100111100011010011011111 \\ & 00111000011101010000100111011111010010100100100 \quad \text{, de longueur } 127 : + 0X^{125} + 0X^{126} \end{aligned}$$

sachant que ce message a été perturbé (par au plus une seule erreur) lors de la transmission.

Information : on a introduit précédemment une fonction (simpliste) qui transforme un polynôme (liste de $N \times 8$ "pseudo-bits") en la chaîne associée (de N caractères du code ASCII 7 bits).

$< \mathscr{FIN} >$ (énoncé du TP Minitel)

7 TP Minitel. Squelette de programme

```
(*-----------------------------------------------------------
 | TP Informatique MP -- Lycée Pothier - Orléans           |  SQUELETTE
 | Fonctionnement du Minitel : Correction d'erreur par CRC |
 | Représentation par polynômes                            |
 -----------------------------------------------------------*)
(* === Edition d'un polynôme de type int list ===*)
let string_of_int_poly f =
  let u = aux 0 f
  where rec aux k = function
      []   -> ""
    | a::q -> let p = aux (k+1) q
              in  if a = 0 then p
                  else let h =  if k = 0 then string_of_int a
                                else   (if a = 1 then ""
                                        else if a = -1 then "-"
                                             else string_of_int a)
                                     ^ (if k= 1 then "X"
                                        else " X^" ^ (string_of_int k))
                       in if string_length p = 0 then h
                          else if p.[0] = '-' then h ^ p
                               else h ^ "+" ^ p
  in if u = "" then "0" else u
;;
(* string_of_int_poly : int list -> string = <fun> *)

let G = [1; 0; 0; 1; 0; 0; 0; 1];;
print_string (string_of_int_poly G);;

(*=================== INCORRECT A PARTIR D'ICI ======================*)

(* === Multiplication par G = 1+X^3+X^7 === *)
let rec multiply_byG = function             (* f -> f x G *)
  | []   -> []
  | a::q -> match multiply_byG q with  .... (* 2 ou 3 lignes à écrire *)
;;
(* multiply_byG : int list -> int list = <fun> *)

let G2 = multiply_byG G;;
print_string (string_of_int_poly G2 );;

(* === Perturbation du coefficient de X^e === *)
let rec perturbe e = function
  | []   -> ...                         (* 1 ligne à écrire *)
  | a::q -> ...                         (* 1 ligne à écrire *)
;;
(* perturbe : int -> int list -> int list = <fun> *)

let G' = G;;
let G' = perturbe 2 G;;
let G' = perturbe 10 G;;

(* === Quotient et Reste dans la division par G = 1+X^3+X^7 === *)
let QRdivide_byG f =            (*  f --> quotient, reste *)
  let rq = aux f
     where rec aux = function  (* traitement des bits un à un *)
       | []   -> []
       | a::q -> match ... with
                 | ... -> ...                          (* 1 ligne à écrire *)
                 | ... -> ...                          (* 1 ligne à écrire *)
  in match rq with              (* séparation en quotient, reste *)
     | ... -> ...                                      (* 1 ligne à écrire *)
     | ... -> ...                                      (* 1 ligne à écrire *)
;;
(* QRdivide_byG : int list -> int list * int list = <fun> *)
```

```
let G2 = [1;0;0;0;0;0;1;0;0;0;0;0;0;0;1];;  (* G x G *)
let H  = [0;1;0;1;0;0;0;0;0;0;0;0;0;0;1];;  (* G x G + 1+X+X^3+X^6 *)
print_string (string_of_int_poly H);;
let q, r = QRdivide_byG H;; print_string (string_of_int_poly q);; print_string (string_of_int_poly r);;

(* === Recherche de l'indice du bit erroné === *)
let bad_indice r = aux 0 [1;0;0;0;0;0;0]      (* Ne se termine que si il y a indice erroné *)
  where rec aux m ( ... as rxm) =
     if rxm = r then ...               (* égalité générique sur listes de même longueur *)
     else ...
;;
(* Warning: this matching is not exhaustive.
   bad_indice : int list -> int = <fun> *)                (* 3 lignes à compléter *)

let H = [0;0;0;0;0;0;0;0;0;0;1];;  (* X^10 *)
let q, r = QRdivide_byG H;;
bad_indice r;;

(*=== TESTS ===*)
(* === Génération d'un polynôme de degré au plus u, à u+1 coefficients === *)
let rec genere_lim = function
       -1 -> []
     | u -> (random__int 2):: (genere_lim (u-1))
;;
(* genere_lim : int -> int list = <fun> *)

(* === génération d'un polynôme de degré au plus u, à v > u coefficients === *)
let rec genere = function                        (* Non sécurisée (pour tests) *)
       -1, v -> list_of_vect (make_vect v 0)
     | u , v -> (random__int 2):: (genere (u-1, v-1))
;;
(* genere : int * int -> int list = <fun> *)

let H = genere_lim 5;; print_string (string_of_int_poly H);;
let H = genere (12,127);;    print_string (string_of_int_poly H);;

(*=====================================================*)
(*    TEST (sur uniquement la partie des 15 octets)     *)
(*=====================================================*)
(* === Enchaînements de test : Codage - Transmission - Décodage correction ===*)
let P = genere (20, 120);;      print_string (string_of_int_poly P);;

let A = multiply_byG P;;        print_string (string_of_int_poly A);;
let A = perturbe 10 A;;         print_string (string_of_int_poly A);;

let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);; print_string (string_of_int_poly R);;

let e = bad_indice R;;
let A = perturbe e A;;

let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);; print_string (string_of_int_poly R);;

P=Q;;

(*=====================================================*)
(*    Complément : codage et décodage de 15 caractères    *)
(*=====================================================*)
(* Chaîne de l'écriture binaire d'une liste, poids forts en tête *)
let rec string_bin_list = function
  | []  -> ""
  | h::q  -> (if h =0 then "0" else "1") ^ (string_bin_list q)
;;
(* string_bin_list : int list -> string = <fun> *)

string_bin_list [1;0;1;0;1;1;1;1];;  (* - : string = "10101111"  *)
```

```
(*-- 1. Codage d'une chaîne vers une liste de bits ---*)
(*------- Transformation d'un char ASCII 7, éditable, en octet liste avec 8 bit de parité *)
let bin_list_of_char = function c ->
  let n = ref (int_of_char c) and p = ref 0 and ln = ref [] in
  if (!n > 126 ) or (!n < 32 )
  then n := 127;                        (* Char non usuel -> [] *)
  for s = 0 to 6
  do let b = ((!n lsr s ) land 1) in
     p := !p lxor b;
     ln := b::!ln
  done;
  !p ::!ln
;;
(* bin_list_of_char : char -> int list = <fun> *)

string_bin_list (bin_list_of_char `A` );;  (* - : string = "01000001" *)
string_bin_list (bin_list_of_char `C` );;  (* - : string = "11000011" *)

(*-------  transformation d'une chaîne de caractères en bin_int_list_of_char *)
let bin_list_of_string = function s ->
  let l = ref [] in
  for i = 0 to string_length s -1
  do l := (bin_list_of_char s.[i]) @ !l done;
  !l
;;
(* bin_list_of_string : string -> int list = <fun>*)

bin_list_of_string "AC";;   (* - : int list = [1; 1; 0; 0; 0; 0; 1; 1; 0; 1; 0; 0; 0; 0; 0; 1] *)

(*-- 2. décodage d'une liste de bits vers chaîne ----*)
(*------- Transformation d'une liste de 8 bit en char *)
let char_of_bin_list l = char_of_int ((aux 7 l) land 127)
  where rec aux k = function
     [] -> 0
    | a::q -> (a lsl k) + aux (k-1) q
;;
(* char_of_bin_list : int list -> char = <fun> *)

char_of_bin_list (bin_list_of_char `A` );;
char_of_bin_list (bin_list_of_char `C` );;

(*------- Transformation d'une liste de N * 8 bit en string *)
let rec string_of_bin_list = function
  |[] -> ""
  | a0::a1::a2::a3::a4::a5::a6::a7::q ->
       let lc = [a0;a1;a2;a3;a4;a5;a6;a7] in
       (string_of_bin_list q) ^ (string_of_char (char_of_bin_list lc) )
;;
(* Warning: this matching is not exhaustive.
   string_of_bin_list : int list -> string = <fun>  *)

string_of_bin_list (bin_list_of_string "AC");;       (* --> "AC" *)

(*=========================================================*)
(*   TEST DE DECODAGE REEL SUR UN MESSAGE DE 15 CARACTERES *)
(*=========================================================*)
(* Message codé recu, de 127 bits (privé du 17 octet et du bit de parité global) *)
let A = [0;1;1;0;0;0;0;1;0;0;1;1;0;0;0;1;0;0;0;1;0;0;0;0;0;0;0;0;1;1;0;0;0;0;0;0;0;0;0;0;
1;1;1;1;0;0;0;0;0;0;0;0;1;1;1;1;1;0;0;1;1;1;1;1;0;0;0;1;1;0;1;0;0;1;1;0;1;1;1;1;1;0;0;1;1;1;
0;0;0;0;1;1;1;0;1;0;1;0;0;0;0;1;0;0;1;1;1;0;1;1;1;1;1;0;1;0;0;1;0;1;0;0;1;0;1;0;0];;

Décodage et correction à écrire ...

(* === FIN (Minitel. Squelette de programme) === *)
```

$< \mathscr{F}\mathscr{I}\mathscr{N} >$ (squelette du TP Minitel)

8 TP Minitel. Programme corrigé

```
(*-------------------------------------------------------------
 | TP Informatique MP -- Lycée Pothier - Orléans               |  CORRIGE
 | Fonctionnement du Minitel : Correction d'erreur par CRC     |
 | Représentation par polynômes                                |
 |   (Voir aussi (old TP) une représentation par big_int ...)  |
 -------------------------------------------------------------*)
(* === Edition d'un polynôme de type int list ===*)
let string_of_int_poly f =
  let u = aux 0 f
  where rec aux k = function
      []   -> ""
    | a::q -> let p = aux (k+1) q
              in  if a = 0 then p
                  else let h =  if k = 0 then string_of_int a
                                else   (if a = 1 then ""
                                        else if a = -1 then "-"
                                             else string_of_int a)
                                     ^ (if k= 1 then "X"
                                        else " X^" ^ (string_of_int k))
                       in if string_length p = 0 then h
                          else if p.[0] = '-' then h ^ p
                               else h ^ "+" ^ p
  in if u = "" then "0" else u
;;
(* string_of_int_poly : int list -> string = <fun> *)

let G = [1; 0; 0; 1; 0; 0; 0; 1];; print_string (string_of_int_poly G);;

(* === Multiplication par G = 1+X^3+X^7 === *)
let rec multiply_byG = function    (* f -> f x G *)
  | []   -> []
  | a::q -> match multiply_byG q with
              | a0::a1::a2::a3::a4::a5::a6::w
                   -> a::a0::a1::(a2+a) mod 2 ::a3::a4::a5::(a6+a) mod 2 ::w
              | _  ->  [a;0;0;a;0;0;0;a]
;;
(* multiply_byG : int list -> int list = <fun> *)

let G2 = multiply_byG G;;  print_string (string_of_int_poly G2 );;

(* === Perturbation du coefficient de X^e === *)
let rec perturbe e = function
    []   -> if e = 0 then [1] else 0::(perturbe (e-1) [])
  | a::q -> if e = 0 then ((a+1) mod 2)::q else a::(perturbe (e-1) q)
;;
(* perturbe : int -> int list -> int list = <fun> *)

let G' = G;;
let G' = perturbe 2 G;;  let G' = perturbe 10 G;;

(* === Quotient et Reste dans la division par 1+X^3+X^7 === *)
let QRdivide_byG f =               (*  f --> quotient, reste *)
  let rq = aux f
  where rec aux = function         (* traitement des bits un à un *)
    | []   -> []
    | a::q -> match a::(aux q) with
                | a0::a1::a2::a3::a4::a5::a6::a7::w
                    -> (a0+a7) mod 2 ::a1::a2::(a3+a7) mod 2 ::a4::a5::a6::a7::w
                | r -> r
  in match rq with                  (* séparation en quotient, reste *)
      | a0::a1::a2::a3::a4::a5::a6::w -> w, [a0;a1;a2;a3;a4;a5;a6]
      | _                             -> [], rq
;;
(* QRdivide_byG : int list -> int list * int list = <fun> *)
```

```
let G2 = [1;0;0;0;0;0;1;0;0;0;0;0;0;0;1];;  (* G x G *)
let H  = [0;1;0;1;0;0;0;0;0;0;0;0;0;0;1];;  (* G x G + 1+X+X^3+X^6 *)
print_string (string_of_int_poly H);;
let q, r = QRdivide_byG H;; print_string (string_of_int_poly q);; print_string (string_of_int_poly r);;

(* === Recherche de l'indice du bit erroné === *)
let bad_indice r = aux 0 [1;0;0;0;0;0;0]
  where rec aux m (a0::a1::a2::a3::a4::a5::a6::w as rxm) =
     if rxm = r then m else aux (m+1) [a6;a0;a1;(a2+a6) mod 2; a3;a4;a5]
;;
(* Warning: this matching is not exhaustive.
   bad_indice : int list -> int = <fun> *)

let H = [0;0;0;0;0;0;0;0;0;0;1];;  (* X^10 *)
let q, r = QRdivide_byG H;;
bad_indice r;;

(* === Génération d'un polynôme de degré au plus u, à u+1 coefficients === *)

let rec genere_lim = function
    -1 -> []
   | u -> (random__int 2):: (genere_lim (u-1))
;;
(* genere_lim : int -> int list = <fun> *)

(* === génération d'un polynôme de degré au plus u, à v coefficients === *)
let rec genere = function
    -1,  v -> list_of_vect (make_vect v 0)
   | u , v -> (random__int 2):: (genere (u-1, v-1))
;;
(* genere : int * int -> int list = <fun> *)

let H = genere_lim 5;;     print_string (string_of_int_poly H);;
let H = genere (12,127);;  print_string (string_of_int_poly H);;

(*=====================================================*)
(*    TEST (sur uniquement la partie des 15 octets)    *)
(*=====================================================*)
(* === Enchaînements de test : Codage - Transmission - Décodage correction ===*)
let P = genere (20, 120);;
print_string (string_of_int_poly P);;

let A = multiply_byG P;; print_string (string_of_int_poly A);;
let A = perturbe 10 A;;  print_string (string_of_int_poly A);;

let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);; print_string (string_of_int_poly R);;

let e = bad_indice R;;
let A = perturbe e A;;

let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);; print_string (string_of_int_poly R);;
P=Q;;

(*=====================================================*)
(*    Complément : codage et décodage de 15 caractères    *)
(*=====================================================*)
(* Chaîne de l'écriture binaire d'une liste, poids forts en tête *)
let rec string_bin_list = function
  | []  -> ""
  | h::q  -> (if h =0 then "0" else "1") ^ (string_bin_list q)
;;
(* string_bin_list : int list -> string = <fun> *)

string_bin_list [1;0;1;0;1;1;1;1];;  (* - : string = "10101111"  *)
```

```
(*-- 1. Codage d'une chaîne vers une liste de bits ---*)
(*------ Transformation d'un char ASCII 7, éditable, en octet liste avec 8 bit de parité *)
let bin_list_of_char = function c ->
  let n = ref (int_of_char c) and p = ref 0 and ln = ref [] in
  if (!n > 126 ) or (!n < 32 ) then n := 127;          (* Char non usuel -> [] *)
  for s = 0 to 6
  do let b = ((!n lsr s ) land 1) in
     p := !p lxor b;
     ln := b::!ln
  done;
  !p ::!ln
;;
(* bin_list_of_char : char -> int list = <fun> *)

string_bin_list (bin_list_of_char `A` );;  (* - : string = "01000001" *)
string_bin_list (bin_list_of_char `C` );;  (* - : string = "11000011" *)

(*-------  transformation d'une chaine de caractères en bin_int_list_of_char *)
let bin_list_of_string = function s ->
  let l = ref [] in
  for i = 0 to string_length s - 1 do l := (bin_list_of_char s.[i]) @ !l done;
  !l
;;
(* bin_list_of_string : string -> int list = <fun>*)

bin_list_of_string "AC";; (* - : int list = [1;1;0;0;0;0;1;1;0;1;0;0;0;0;1] *)

(*-- 2. décodage d'une liste de bits vers chaîne ----*)
(*------- Transformation d'une liste de 8 bit en char *)
let char_of_bin_list l = char_of_int ((aux 7 l) land 127)
  where rec aux k = function
     [] -> 0
    | a::q -> (a lsl k) + aux (k-1) q
;;
(* char_of_bin_list : int list -> char = <fun> *)

char_of_bin_list (bin_list_of_char `A` );;  char_of_bin_list (bin_list_of_char `C` );;

(*------- Transformation d'une liste de N * 8 bit en string *)
let rec string_of_bin_list = function
   [] -> ""
  | a0::a1::a2::a3::a4::a5::a6::a7::q -> let lc = [a0;a1;a2;a3;a4;a5;a6;a7] in
                                        (string_of_bin_list q) ^ (string_of_char (char_of_bin_list lc))
;;
(* string_of_bin_list : int list -> string = <fun>  *) (* Warning: this matching is not exhaustive. *)

string_of_bin_list (bin_list_of_string "AC");;   (* --> "AC" *)

(*===============       TEST    ==================*)
let P = bin_list_of_string "Test TP minitel";; print_string (string_of_int_poly P);;
let A = multiply_byG P;;                       print_string (string_of_int_poly A);;
let A = perturbe 87 A;;                        print_string (string_of_int_poly A);;
string_bin_list A;;
let A = 0;1;1;0;0;0;0;1;0;0;1;1;0;0;0;1;0;0;0;1;0;0;0;0;0;0;0;0;1;1;0;0;0;0;0;0;0;0;0;0;
1;1;1;1;0;0;0;0;0;0;0;0;0;1;1;1;1;0;0;1;1;1;1;1;0;0;0;1;1;0;1;0;0;1;1;0;1;1;1;1;1;0;0;1;1;1;
0;0;0;0;1;1;1;0;1;0;1;0;0;0;0;1;0;0;1;1;1;0;1;1;1;1;1;0;1;0;0;1;0;1;0;0;1;0;1;0;0];;
let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);;  print_string (string_of_int_poly R);;
let e = bad_indice R;;      let A = perturbe e A;;
let Q, R = QRdivide_byG A;; print_string (string_of_int_poly Q);;  print_string (string_of_int_poly R);;
P = Q;;
string_of_bin_list Q;;

(* === FIN (Minitel. Programme corrigé) === *)
```

$<$ $\mathscr{FIN}$ $>$ (Corrigé du TP Minitel)