DM - Épisode 1

Ceci est la première partie d'un DM qui sera composé de deux parties. Cette première partie a pour but de vous familiariser avec la problématique générale, les notations et la bibliothèque GMP. Vos travaux doivent être envoyés à ***sylvain.chevillard@ens-lyon.fr*** *au plus tard le dimanche 22 mars 2009 à 23h59. Les retards seront évidemment sanctionnés.*

Dans ce devoir nous allons étudier plusieurs aspects du problème de la multiplication par une constante. La formulation générale est la suivante : soit M une constante entière. On cherche à écrire un algorithme qui prend en entrée un entier x et qui renvoie en retour le produit Mx. La connaissance préalable de M permet d'écrire un algorithme plus efficace que la simple utilisation de l'algorithme de multiplication générique avec M et x comme arguments.
Les différents exercices de ce DM vous demanderont d'écrire des algorithmes pour synthétiser du code C : autrement dit nous vous demandons d'écrire un programme `P` qui prend en argument un entier M et qui renvoie en sortie (par exemple sur la sortie standard ou dans un fichier) le code-source en langage C d'un programme `MulByM`. Votre programme `P` pourra être écrit dans le langage de votre choix parmi C, OCaml ou Python. Si vous tenez absolument à utiliser un autre langage contacter votre TDman qui examinera votre cas.
Le programme `MulByM` doit être écrit en C et utiliser la bibliothèque `GMP`[1]. Par exemple, si on appelle votre programme `P` avec $M = 5$, il pourrait produire le code suivant :

```
#include <gmp.h>

void MulBy5(mpz_t result, mpz_t x) {
  mpz_set(result, x);
  mpz_mul_2exp(result, x, 2);  /* result = 4x */
  mpz_add(result, result, x);  /* result = 5x */

  return;
}
```

Dernière remarque : vos programmes `P` devront être capables de prendre en argument un entier M de taille arbitraire (par exemple, si vous écrivez en C, utilisez `GMP`[2] ; en OCaml, utilisez `Big_int` ; en Python, c'est supporté nativement).
Chaque exercice vous demandera d'écrire un programme `P` différent. Vous devrez envoyer toutes les sources de vos programmes à **sylvain.chevillard@ens-lyon.fr** accompagnées des éventuelles instructions nécessaire à la compilation et l'exécution de vos fichiers. Lorsqu'il y a lieu, on attend des commentaires dans vos sources ou sous forme de fichier séparé joint. Le correcteur appréciera que le code que vous synthétisez soit lisible, indenté, et annoté (comme dans l'exemple précédent avec les commentaires indiquant la valeur d'une variable à un moment de l'algorithme).

[1] `GMP` est très facile à utiliser. Lisez tout particulièrement http ://gmplib.org/manual/GMP-Basics.htmlGMP-Basics, http ://gmplib.org/manual/Integer-Functions.htmlInteger-Functions, et plus généralement (téléchargement, installation, manuel en PDF, etc.) http ://gmplib.org/

[2] si vous préférez programmer en C++, jetez un coup d'œil à http ://gmplib.org/manual/C_002b_002b-Interface-General.html. En revanche, attention, on demande à ce que les programmes `MulByM` que vous générez soient écrits en C et pas en C++.

Question 1.
Écrire un programme qui prend en argument un grand entier M et renvoie la liste (ou le tableau) constitué des bits de son développement binaire.
Écrire un programme `AlgoNaif` qui prend en argument un entier M et synthétise un programme C de multiplication par M, par l'algorithme naïf d'additions/décalages (comme dans l'exemple présenté plus haut avec $M = 5$).

Question 2.
Écrire un programme qui prend en argument une liste L (ou un tableau) de bits (représentant un entier M) et qui calcule une liste L' (ou un tableau) telle que L' est le *recodage de Booth modifié* de L.
Écrire un programme `AlgoBooth` qui prend en argument un entier M et synthétise un programme C de multiplication par M, par l'algorithme naïf d'addition, soustraction et décalage (donc, s'appuyant sur le recodage de Booth modifié).