TD/TP 1: Librairie BIGNUM

Enoncé :

La taille des entiers utilisés dans le langage C est limitée. Le type entier le plus grand, `unsigned long long`, est codé sur 64 bits et permet donc de representer des entiers compris entre 0 et $2^{64} - 1 = 18446744073709551615$.

Neamoins, certaines applications (*e.g.,* calcul haute précision, fonctions de hachage) ont besoin d'utiliser des entiers de taille plus grande. Le but de ce TP est donc d'implementer une librairie BIGNUM permettant de travailler sur des entiers de taille 512 bit.

Les operations de base sur les entiers sont :

- addition, soustraction
- multiplication
- division entière
- modulo

Pour representer ces entiers de grande taille, on utilisera la structure suivante :

```
#define TAILLE 16
typedef struct {
        int num[TAILLE];
} bignum;
```

Votre code source devra comporter un fichier `bignum.h` contenant la structure définie ci-dessus ainsi que les signatures de fonctions `somme`, `multiplication`, `soustraction`, `division_entiere` et `modulo`. Chacune de ces fonctions prend en paramètre deux structures bignum et retourne un bignum résultat de l'operation.

Le fichier `bignum.c` contiendra entre autres l'implementation des fonctions définies dans `bignum.h`.

Enfin, un fichier `main.c` contiendra la fonction `main()` servant à tester votre librairie BIGNUM.

Question 1

Coder la fonction `int retourner_bit(bignum num, int position)` qui retourne la valeur du bit qui se situe à la position `position` dans le bignum `num`.

Question 3

À l'aide de la fonction `retourner_bit`, coder la fonction `void affichage_binaire(bignum num)` qui affiche à l'écran la represetation binaire de `num`. Tester la fonction `retourner_bit` et `affichage_binaire` dans votre main().

Question 2

À l'aide de la fonction `retourner_bit`, coder la fonction `int comparer(bignum n1, bignum n2)` qui compare bit à bit deux entiers bignum n1 et n2. Elle retourne 0 si $n1 = n2$, -1 si $n1 < n2$ et 1 si $n1 > n2$.

Question 4

Coder la fonction `void affecter_bit(bignum num, int position, int valeur)` qui affecte la valeur `valeur` au bit qui se situe à la position `position` dans l'entier `num`.

Question 5

En utilisant la fonction `affecter_bit`, coder la fonction `void initialiser(bignum num)` qui initialise tous les bits de bignum à zero.

Question 6

En utilisant la fonction `retourner_bit`, coder deux fonctions :

- `int somme_bit(bignum n1, bignum n2, int position, int retenue)` qui renvoie la somme de 3 bits : le bit de n1 situé à la position `position`, le bit de n2 situé à la position `position` et la retenue.
- `int calcul_retenue(bignum n1, bignum n2, int position, int retenue)` qui calcule la valeur de la retenue résultant de l'opération précedente.

Question 7

En utilisant les fonctions codées dans les questions 2 et 3, coder la fonction `somme` qui renvoie la somme de deux entiers bignum.

Question 8

À l'aide de la fonction `somme`, coder la fonction `multiplication`.

Question 9

En s'inspirant de la question 4, coder les fonctions `soustraction_binaire` et `retenue_binaire` qui retournent respectivement le résultat de la soustraction binaire et la valeur de la retenue binaire.

Question 10

À l'aide de la question précedente et de `affecter_bit`, coder la fonction `soustraction`.

Question 11

À l'aide des fonctions `soustraction` et `comparer`, coder la fonction `division_entiere`.

Question 12

En utilisant les fonctions `division_entiere` et `soustraction`, coder la fonction `modulo`