# TD 5

Sur les machines classiques, le type int peut représenter des entiers sur 32 bits soit de $-2^{31}$ à $2^{31}-1$ ($2^{31} \equiv 2$ milliards). Pour utiliser des nombres entiers plus grands, l'utilisateur doit définir par logiciel le traitement sur ces "grands nombres" appelés les **bignums**. Ce TD propose d'implanter la structure pour le type bignum et les opérations de bases (addition, soustraction, multiplication) travaillant sur ce type.

On utilisera les types suivants :

```
/* Taille maximum d'un bignum */
#define TAILLE_MAX 10

/* Signe + / - */
typedef enum {PLUS = 0,MOINS = 1} signe;

typedef unsigned short int chiffre;

typedef struct bignum {
  int longueur;                 /* Le nombre de chiffres significatifs */
  int taille;                   /* nombre maximal de chiffre pour ce bignum */
  signe signe;                  /* Le signe du bignum */
  chiffre num[TAILLE_MAX];      /* La représentation du bignum */
} bignum_t;
```

Les bignums seront représentés par la structure `bignum`. Le champ `longueur` indique le nombre de chiffres significatifs. Le champs `taille` indique l'espace mémoire réservé pour le nombre. Le champ `signe` indique le signe (`PLUS/MOINS`) du nombre. Enfin le champ `num` contient tous les chiffres du bignum tel que num[i] soit le chiffre correspondant à son rang (unité en 0, dizaine en 1, ...).

********** Correction **********

```
/* Retourne une structure initialisée.
   Fonction utilisée si une structure bignum est mal définie */
bignum_t bignum_err() {
  bignum_t res = initialiser();
  return res;
}

/* Vérifie la bonne définition d'une structure bignum */
bool_t est_def(bignum_t num) {
  return ((num.longueur != 0) && (num.longueur != -1));
}
```

1. Ecrivez une fonction appelée `initialiser` qui renvoie une structure avec l'ensemble des valeurs de la structure `bignum` initialisées. On initialisera la longueur à -1.

********** Correction **********

```
bignum_t initialiser() {
  bignum_t num;
  num.longueur = -1;
  num.taille = TAILLE_MAX;
  num.signe = PLUS;

  int i = 0;
  while (i < TAILLE_MAX) {
    num.num[i] = 0;
    i++;
  }
  return num;
}
```

2. Ecrivez une fonction appelée `lire` qui renvoie la structure bignum lue au clavier. Cette fonction devra lire au clavier un signe ('+' ou '-' obligatoirement présent) suivit d'une suite de chiffres compris entre '0' et '9' pour les mettre dans la structure bignum. On utilisera un tableau temporaire pour réaliser l'inversion des chiffres après les avoir lu du clavier. On utilisera `getchar()` pour lire les caractères du clavier.

```
Exemple : Clavier : -9456 donne
struct bignum {
  longueur = 4;
  signe = MOINS;
  num = {6, 5, 4, 9};
}
```

********** Correction **********

```
/* Lit un bignum sur le terminal */
bignum_t lire() {

  int c, i;
  chiffre temp[TAILLE_MAX];     /* Pour l'inversion */
  bignum_t num = initialiser();;

  num.longueur = 0;

  /* Lit le signe */
  c = getchar();
  switch (c) {
  case '+':
    num.signe = PLUS;
    break;
```

```
  case '-':
    num.signe = MOINS;
    break;
  default:
    printf("Il me manque le signe.\n");
    return bignum_err();
  }

  /* Lit tous les caracteres jusqu'a la fin de ligne */
  while ((c = getchar()) != '\n') {
    if (num.longueur < TAILLE_MAX && c >= '0' && c <= '9') {
      temp[num.longueur] = c - '0';
      num.longueur++;
    }
    else {
      printf("Depassement de capacite ou caractere incorrecte\n");
      return bignum_err();
    }
  }

  /* Inverse temp dans num[] */
  for (i = 0; i < num.longueur; i++)
    num.num[i] = temp[num.longueur - i - 1];

  return num;
}
```

3. Ecrire une fonction appelée `ecrire` qui affiche la valeur d'un bignum sur l'écran.

********** Correction **********

```
/* Affiche le bignum sur le terminal */
void ecrire(bignum_t num) {

  if (est_def(num)) {
    /* Affiche le signe */
    switch (num.signe) {
    case PLUS:
      printf("+");
      break;
    case MOINS:
      printf("-");
      break;
    default:
      printf("Erreur de signe !\n");
    }
```

```
    /* Affiche chaque chiffre en terminant par l'unite */
    int i;
    for (i = num.longueur - 1; i >= 0; i--) {
      printf("%d", num.num[i]);
    }
    printf("\n");
  }
  else {
    printf("Pas def\n");
  }
}
```

4. Ecrire une fonction `neg` qui prend en argument un bignum et qui retourne son opposée.

********** Correction **********

```
/* Retourne l'opposée du bignum donnée en paramètre */
bignum_t neg(bignum_t num) {
  bignum_t neg = initialiser();

  if (est_def(num)) {
    neg = num;
    /* On inverse le signe dans b */
    if (num.signe == PLUS) neg.signe = MOINS;
    else neg.signe = PLUS;
  }
  return neg;
}
```

5. Ecrire la fonction `addition_aux` pour *des bignums de même signe*. Cette fonction doit vérifier que les deux bignums ont le même signe pour faire le calcul et doit renvoyer une erreur sinon.

********** Correction **********

```
/* additionne a et b (de meme signe) */
bignum_t addition_aux (bignum_t a, bignum_t b) {
  bool_t res;
  bignum_t c = initialiser();
  int som,              /* somme de 2 chiffres */
    lgmax,              /* plus grande longueur entre a et b */
    val_a, val_b,       /* valeur des chiffres de a et b */
    i,                  /* compteur de boucle */
    reste = 0;          /* reste a propager */
```

```
  c.longueur = 0;

  if ((!est_def(a)) || (!est_def(b))) res = FALSE;
  else if (a.signe == b.signe) {
    res = TRUE;

    /* Longueur MAX */
    if (a.longueur > b.longueur) lgmax = a.longueur;
    else lgmax = b.longueur;

    /* Pour tous les chiffres */
    for (i = 0; i < lgmax; i++) {
      /* Prend la valeur du chiffre ou 0 si on a depasse la longueur du plus
         petit chiffre */
      if (i >= a.longueur) val_a = 0;
      else val_a = a.num[i];
      if (i >= b.longueur) val_b = 0;
      else val_b = b.num[i];

      /* Calcul la somme plus la retenue */
      som = val_a + val_b + reste;
      if (som >= 10) {    /* On a une retenue */
        som -= 10;
        reste = 1;
      }
      else  reste = 0;    /* Pas de retenue */

      c.num[i] = som;  /* On a le ieme chiffre */
    }

    if (reste != 0) {
      if (lgmax < TAILLE_MAX) {
        c.num[lgmax] = 1;
        c.longueur = lgmax + 1;
      }
      else res = FALSE;         /* le bignum c serait trop grand */
    }
    else c.longueur = lgmax;

    c.signe = a.signe;
  }

  if (res == TRUE) return c;
  else return bignum_err();
}
```

6. Ecrire la fonction `soustraction_aux` pour *des bignums de même signe*.

********** Correction **********

```
/* a > b ? */
bool_t compare (bignum_t a, bignum_t b) {
  bool_t res;                   /* Le resultat */
  int i = a.longueur - 1;       /* indice de parcours du bignum */

  /* Les deux nombres ne sont pas definis */
  if ((!est_def(a)) || (!est_def(b)))  res = FALSE;
  else if (a.signe != b.signe)
    /* signe PLUS > signe MOINS */
    res = (a.signe == PLUS ? TRUE : FALSE);
  else if (a.longueur == b.longueur)
    /* Compare en terminant par les unitees */
    do {
      res = (a.num[i] > b.num[i]);
      i--;
    } while(a.num[i + 1] == b.num[i + 1] && i > 0);
  else
    /* Le plus long est le plus grand  si a et b sont positifs*/
    res = (a.signe == PLUS ?
           a.longueur > b.longueur :
           a.longueur < b.longueur);

  return res;
}

/* Soustraction */
bignum_t soustraction_aux (bignum_t a, bignum_t b) {
  bignum_t val_max,             /* Le plus grand des bignum a ou b */
    val_min,                    /* Le plus petit */
    n_a, n_b,                   /* neg de a et b */
    c = initialiser();          /* Le resultat */
  int i,                         /* Compteur de boucle */
    reste = 0,                   /* Le reste */
    sous;                        /* soustraction de 2 chiffres */

  c.longueur = 0;

  if ((!est_def(a)) || (!est_def(b)))
    return(bignum_err());
  if (a.signe != b.signe)
    return(bignum_err());

  /* (-a) - (-b) c'est (+b) - (+a) */
  if (b.signe == MOINS) {
    n_a = neg(a);
```

```
  n_b = neg(b);
  return soustraction_aux(n_b, n_a);
}

/* On ne sait soustraire a-b que si a>=b, on doit donc faire
   val_max-val_min */
if (!compare(b, a)) {
  val_max = a;
  val_min = b;
  c.signe = PLUS;
}
else {
  val_max = b;
  val_min = a;
  c.signe = MOINS;
}

/* Pour tous les chiffres */
for(i = 0; i < val_max.longueur; i++) {
  /* si on depasse la longueur du plus petit on veut y trouver des 0 */
  if (i >= val_min.longueur) val_min.num[i] = 0;
  if (val_max.num[i] >= val_min.num[i]) {
    sous = val_max.num[i] - (val_min.num[i] + reste);
    reste = 0;
  }
  else {  /* On a une retenue */
    sous = (val_max.num[i] + 10) - (val_min.num[i] + reste);
    reste = 1;
  }

  c.num[i] = sous;
}

i = val_max.longueur - 1;

/* Enleve les zeros devant le nombre : 99-90 donne 09, on veut 9 */
while ( i >= 0 && !(c.num[i--]) );
c.longueur = i + 2;

return c;
}
```

7. Améliorer les fonctions `addition` et `soustraction` pour traiter le cas des 2 arguments de signes différents.

********** Correction **********

```
/* Addition de bignums quelconques (signes differents ) */
bignum_t addition (bignum_t a, bignum_t b) {
  bignum_t m_a, m_b, c;

  if ((!est_def(a)) || (!est_def(b)))
    return bignum_err();

  if (a.signe == b.signe)
    return addition_aux(a, b);
  else if (a.signe == MOINS) {
    m_a = neg(a);
    if (compare(m_a, bignum_err()))
      return m_a;
    c = soustraction_aux(b, m_a);
  }
  else {
    m_b = neg(b);
    if (compare(m_b, bignum_err()))
      return m_b;
    c = soustraction_aux(a, m_b);
  }
  return c;
}

bignum_t soustraction (bignum_t a, bignum_t b) {
  bignum_t m_b, c;

  if ((!est_def(a)) || (!est_def(b)))
    return bignum_err();

  if (a.signe == b.signe) c = soustraction_aux(a, b);
  else {
    /* (-a) - (+b) c'est (-a) + (-b) et
       (+a) - (-b) c'est (+a) + (+b) donc
       il faut inverser b pour passer a l'addition */
    m_b = neg(b);
    if (!est_def(m_b))
      return m_b;
    c = addition_aux(a, m_b);
  }
  return c;
}
```

8. Ecrire une fonction `puissance10` qui prend un bignum et un entier en argument. Cette fonction doit calculer bignum*$10^{entier}$.

********** Correction **********

```
/* calcul a*10^n */
bignum_t puissance10 (bignum_t a, int n) {
  int i, j = 0;
  bignum_t b = initialiser();

  b.longueur = 0;

  if ((!est_def(a)) || n < 0 || (a.longueur + n) >= TAILLE_MAX)
    return bignum_err();

  /* Pour tous les chiffres de a plus les n 0 qu'on va ajouter */
  for (i = 0; i < a.longueur + n; i++) {
    /* Si on a ajoute les n 0 on recopie a dans b */
    if (i >= n)
      b.num[i] = a.num[j++];
    else
      b.num[i] = 0;
  }
  b.longueur = a.longueur + n;
  b.signe = a.signe;

  return b;
}
```

9. Ecrire la fonction `multiplication`. Vous utiliserez la fonction `addition` et/ou la fonction `puissance10`.

********** Correction **********

```
bignum_t multiplication (bignum_t a, bignum_t b) {
  bignum_t add,                 /* Resultat de a * 1 chiffre de b */
    tmp1, tmp2;                 /* bignums temporaires */
  int i, j;
  bignum_t res = initialiser(); /* Resultat de la multiplication */

  if ((!est_def(a)) || (!est_def(b)))
    return bignum_err();

  res.longueur = 1;
  res.signe = PLUS;
  res.num[0] = 0;

  /* Pour tous les chiffres de b */
  for (i = 0; i < b.longueur; i++) {
    /* Calcul a * chiffre de b dans add*/
```

```
    /* add = bignum zero */
    add.longueur = 1;
    add.signe = PLUS;
    add.num[0] = 0;

    for (j = 0; j < b.num[i]; j++) {
      if (a.signe == PLUS) {
tmp1 = addition(add, a);
        if(!est_def(tmp1))
          return tmp1;
      }
      else {
tmp2 = neg(a);
        if(!est_def(tmp2))
          return tmp2;
tmp1 = addition(add, tmp2);
        if(!est_def(tmp1))
          return tmp1;
      }
      add = tmp1;
    }
    tmp1 = puissance10(add, i); /* ajoute des 0 a add */
    if (!est_def(tmp1))
      return tmp1;
    add = tmp1;
    tmp2 = addition(res, add);  /* ajoute add dans res */
    if (!est_def(tmp2))
      return tmp2;
    res = tmp2;
  }
  /* On a fait la multiplication en valeur absolue, maintenant on met le
     signe */
  if (a.signe != b.signe)
    res.signe = MOINS;

  return res;
}
```

10. (facultatif) Ecrire la fonction `division` entière qui divise un bignum par un autre.