

**Projet du langage c**

**GRANDS ENTIERS NATURELS**



Charly Darcy BYMBOUTSA MOUNDOUNGA

**Responsable : GRANET Vincent**

**Objectif**: Définir et manipuler des grands entiers naturels.

# Introduction

Étant donné quand C le plus grand entier que l’on peut manipuler est le type unsigned long long int qui est représenté en général sur 8 octets sur les PC actuels. Celui-ci n’étant pas le format approprié pour notre étude car nous voulons représenter des nombres dont les formats sont encore plus grand. Alors cela m’a conduit à chercher comment faire pour manipuler des nombres dont le format n’existe pas dans le C de base ?

# Procédé

Pour manipuler en C des entiers bien plus grand que le type unsigned long long int qui est la plus grande représentation de base, j’ai décidé d’utiliser une suite de caractère appelé char dans le langage c. Donc nous allons tout au long de ce projet utiliser cette représentation pour manipuler nos BigNat. Cela m’a conduit à créer un type prédéfini nommé BigNat qui sera utile pour pouvoir créer chacune des fonctions qui sont dues aux différents tests à réaliser.

# Code

Tout d’abord on crée une constance préprocesseur en y insérant un MAX 200.

Cela nous conduit ensuite à faire une déclaration du typedef qui contiendra notre BigNat que nous pourrons utiliser afin de manipuler correctement nos BigNat, dans celle-ci il est question que notre BigNat est bien une chaîne de caractère. Ainsi ayant mis en place notre type BigNat nous pouvons dès à présent commencé à implémenter nos différentes fonctions :

* **Initialisation**

Il est question ici d’implémenter la fonction qui va permettre de faire l’initialisation de nos BigNat que ça soit à 0, à partir d'une chaîne de caractères, à partir d'un entier naturel ou encore à partir d’un autre BigNat.

* **init\_int()**

Dans cette fonction j’ai commencé à faire de l’allocation dynamique, j’ai alloué une chaîne de caractère de taille MAX + 1. Comme il faut toujours qu’une chaîne se termine par un caractère de fin de chaîne (‘\0’) donc on met que la valeur maximal de notre BigNat b est bien égal au caractère de fin de chaîne. On parcourt toute la chaîne et on met chacun des caractères à 0 ou encore on les initialise tous à 0, mais ici ce n’est pas le 0 c’est sa valeur dans la table ASCII. La logique de chacune des opérations étant fait pour être réaliser de la droite vers la gauche on fait l’alignement de chacun de nos caractères, afin de nous préparer à faire les différentes opérations. Cela ce fait car on part de l’avant dernier caractère, on met le dernier caractère ou encore le dernier chiffre grâce au modulo de 10 et quand on le divise par 10 on passe à la case précédente de la chaîne ce sera ainsi jusqu’à ce que a sera différent de 0.

* **real\_len()**

Cette fonction a été implémentée spécialement pour renvoyer la longueur réelle du nombre qui est rentré dans notre chaîne de caractère.

* **init\_str()**

On initialise le BigNat avec une chaîne de caractère et pour ce faire on fait une vérification pour savoir si la longueur de la chaîne de caractère rentré est bien soit inférieur ou égal à la valeur MAX qui est celle de la longueur maximale qui nous est permis d’entrée. Si la longueur de notre chaîne correspond bien alors on initialise un BigNat qu’on mettra bien à 0. On déclare une variable qu’on initialise à MAX – 1, on fait une boucle de telle sorte que si la longueur de notre chaîne rentré - 1 soit supérieur ou égal à 0 alors on continue à décrémenté et pendant ce temps chaque fois on fait une copie de la chaîne tout à droite de notre BigNat. Quand on aura bien fini de vérifier notre condition et qu’on aura alors bien copié toute la chaîne qui se trouve à droite du BigNat alors on va juste retourner notre chaîne.

* **init\_BigNat()**

Cette fonction nous permet de faire l’initialisation d’un BigNat avec un autre BigNat.

Pour cela nous commençons par initialiser un BigNat b qu’on initialise avec l’entier 0.

On fait une boucle for jusqu’à ce que le test x < MAX soit faux pour sortir de la boucle et pendant tout ce temps que la boucle est vraie on met la valeur de notre BigNat a égal dans notre BigNat b.

* **Écriture**
* **ecrire()**

Dans cette procédure nous passons un BigNat et nous faisons une première boucle afin de parcourir tous les éléments qui sont inférieur à notre MAX, on fait une condition afin de récupérer tous les caractères sauf les 0 à gauche. Donc nous procédons ainsi afin de détecter le premier caractère de notre BigNat qui est différent de 0 une fois que c’est fait on fait notre seconde boucle en partant du premier caractère différent de 0 on parcourt toute la chaîne et on les affiches tous les uns à la suite des autres.

* **Comparaison**
* **sup()**

Nous commençons à parcourir avec notre boucle notre chaîne jusqu’à ce que le test soit faux c’est-à-dire que x soit égal à MAX et si on se rend compte que notre BigNat a est plus grand que le BigNat b alors on renvoie **true** pour nous montrer que la condition posé est bien la bonne. Sinon si ce n’est pas le cas on return false car le rôle de notre fonction est de nous donner false lorsque notre BigNat a n’est plus supérieur à b.

* **eq()**

On parcourt tous les éléments de la boucle jusqu’à ce que x soit égal à MAX, et comme le test de la boucle est vrai on fait un test sur le BigNat a et le BigNat b si le test est vrai c’est-à-dire que les deux sont bien différents on return alors false et on continue jusqu’à ce que les deux soit égales ainsi on retourne true pour montrer que nous fonction est bien valide.

* **dif()**

De même on parcourt tous les éléments de la boucle jusqu’à ce que x soit égal à MAX, et comme le test de la boucle est vrai on fait un test sur le BigNat a et le BigNat b si le test est vrai c’est-à-dire que les deux sont bien différents on return alors true et on continue jusqu’à ce que les deux soit différentes ainsi on retourne false pour montrer que nous fonction est bien valide.

* **inf()**

On fait un test sur l’égalité du BigNat a et du BigNat b et si celui-ci s’avère vrai alors on return false et si c’est faux alors on return que le BigNat a est bien inférieur au bigNat b qui est traduit par cela dans le code  .

* **infeq()**

Ayant déjà complémenté la plupart des fonctions servant à faire des comparaisons nous allons directement utiliser celle-ci donc on va faire directement un return de BigNat a inférieur au BigNat b traduit ainsi dans le code .

* **supeq()**

On fait directement un return de BigNat a supérieur au BigNat b traduit ainsi dans le code .

* **Addition**
* **addi()**

On commence par faire un test afin de vérifier que la longueur réelle du BigNat a et du BigNat b sont bien inférieur à la longueur maximale que nous pouvons utiliser. On initialise notre BigNat r et on déclare notre variable n. la boucle for nous permet de nous placer sur le premier caractère différent du caractère de fin de chaîne (MAX – 1). Ensuite on décale vers la gauche en faisant une décrémentation pour pouvoir prendre en compte toute la chaîne et pour cela chaque fois on additionne nos BigNat a et b. On ajoute une retenue lorsque cela s’avère nécessaire c’est-à-dire qu’on ajoute r[x] qui vaut 1 à la prochaine addition de nos BigNat si lors de l’addition précédente le résultat obtenu est supérieur ou égal à 10(n > 9), sinon on ne la prend pas en compte lorsque l’addition de nos BigNat a et b précédente nous donne une valeur qui est inférieur à 10 (n < 9) alors r[x] vaut tout simplement 0. Et à chaque fois nous devons soustraire 3 \* ‘0’ à nos additions car nous travaillons sur des caractères et non sur des chiffres, le 3 c’est parce que nous avons trois opérande et le ‘0’ on utilise son code ASCII. Ces opérations vont se faire jusqu’à ce que nous soyons une case avant la fin de la chaîne pour prendre en compte la retenue éventuelle s’il y en a.  Ici on donne le code ASCII de notre retenue.

* **Soustraction**
* **sous()**

On fait un test qui doit toujours être vrai pour pouvoir faire notre soustraction comme il se doit et celui-ci est qu’il faut la longueur réelle du BigNat a et du BigNat b soit inférieur à longueur MAX et aussi pour faire notre soustraction il faut toujours que le BigNat b soit inférieur au BigNat a. Je fais l’initialisation du BigNat r avec l’entier 0, du BigNat c à partir du BigNat b et de l’entier n. La boucle for nous permet de nous placer sur le premier caractère différent du caractère de fin de chaîne (MAX – 1). Ensuite on décale vers la gauche en faisant une décrémentation pour pouvoir prendre en compte toute la chaîne et pour cela chaque fois on soustrait nos BigNat a et c. Si le BigNat a est supérieur ou égal à 0 au BigNat c C’est-à-dire que **n > 0** alors on n’a pas de retenu dans la soustraction suivante  on prend alors notre résultat de notre soustraction que nous ajoutons le code ASCII de ‘0’ directement. Sinon si le BigNat a est inférieur au BigNat b c’est-à-dire que **n < 0** alors on a bien une retenu qu’on ajoute sur notre b suivant ainsi on a  notre retenu plus n avec en plus le code ASCII de ‘0’. Cette opération s’effectuera ainsi jusqu’à ce que nous soyons une case avant la fin de la chaîne pour prendre en compte la retenu éventuelle s’il y’en a.

* **Multiplication**
* **mult()**

Pour faire la multiplication on fait un test qui doit toujours être vrai et celui-ci est qu’il faut que la longueur réelle du BigNat a plus la longueur réelle du BigNat b soit strictement inférieur ou égal à la longueur MAX. Je fais l’initialisation des BigNat m, r, c et de l’entier n. La première boucle for permet de nous placer sur le premier caractère différent du caractère de fin de chaîne (MAX – 1). Ensuite on décale vers la gauche en faisant une décrémentation pour pouvoir prendre en compte toute la chaîne et pour cela chaque fois on initialise r. Ainsi la deuxième boucle for permet de nous placer sur le premier caractère différent du caractère de fin de chaîne (MAX – 1). Ensuite on décale vers la gauche en faisant une décrémentation pour pouvoir prendre en compte toute la chaîne et pour cela chaque fois on multiplie le BigNat a et le BigNat b si n ≥ 10 alors on a une retenu c[y] qui s’ajoute à la multiplication suivante et si n < 10 alors on n’a pas de retenue ce qui veut dire que c[y] vaut 0. Cette opération s’effectuera ainsi jusqu’à ce que nous soyons une case avant la fin de la chaîne pour prendre en compte la retenu éventuelle s’il y’en a. Pour finaliser notre multiplication nous devons faire l’addition des différents résultats obtenues avec chacun des caractères de notre BigNat a et le résultat de cette addition nous donne alors notre multiplication de notre BigNat a avec notre BigNat b.

* **Modulo**
* **factorielle()**

On initialise notre BigNat t à l’entier 0 et notre BigNat r à l’entier 1. Si le BigNat a est égal au BigNat t alors on libère la mémoire alloue pour le BigNat t et on affiche notre factorielle qui est égal à 1. Sinon on initialise un BigNat s qu’on initialisera avec la soustraction du BigNat a et du BigNat r, on initialise aussi un BigNat m qui va nous donner multiplication du BigNat a avec le factorielle de la soustraction du BigNat a par le BigNat r. on libère ensuite les mémoires utilise pour les BigNat t, r et s et on return le résultat de notre multiplication qui a été effectué sur m.

* **main()**

Fonction principale de notre programme qui nous permet chacun des tests de nos différentes fonctions en passant à chacune d’entre elles les paramètres qui se doit.