Université de Bordeaux
Année 2022–2023

Licence Informatique, 3e année

Techniques Algorithmiques et Programmation

TP noté – 13 avril 2023 – durée : 2h40

Opérations arithmétiques sur grands nombres entiers

Consignes

Vous avez le droit de consulter une seule ressource sur Internet : les notes de cours. Vous pouvez utiliser vos notes personnelles de cours et de TD, ainsi que vos programmes. C'est une épreuve **individuelle**, vous n'avez pas le droit de communiquer avec vos voisins, proches ou lointains.

La notation prendra principalement en compte la correction de votre code, c'est-à-dire, s'il passe les tests avec succès, et de façon moindre :

- sa lisibilité (présentation et commentaires),
- ses performances, que vous pouvez tester avec la commande `time`,
- l'absence de fuite mémoire, que vous pouvez détecter `valgrind`.

Votre code doit pouvoir être compilé sans erreur ni avertissement du compilateur.

1 Objectif du TP noté

Il s'agit de coder les algorithmes d'addition, de soustraction et de multiplication de grands nombres entiers, dans une base fixée entre 2 et 10. Les algorithmes d'addition et de soustraction sont les algorithmes naïfs (opérations chiffre par chiffre, en propageant des retenues éventuelles). On demande aussi de programmer deux algorithmes de multiplication vus en cours, tous deux récursifs et basés sur la technique « diviser pour régner » : l'algorithme utilisant une récursivité naïve en $O(n^2)$ et l'algorithme de Karatsuba qui économise une multiplication, dont la complexité est $O(n^{\log_2(3)}) \approx O(n^{1.59})$. Certains éléments de ces algorithmes sont rappelés en section 6.

2 Ce que vous devez télécharger et déposer sur Moodle

1. Téléchargez et décompressez l'archive `tp.tgz` depuis Moodle.
2. Éditez **uniquement** le fichier `tp.c`.
3. En fin d'épreuve, déposez le fichier `tp.c` sous Moodle, et uniquement lui, non compressé.

2.1 Ce qui est fourni

2.1.1 Fichiers sources C et fichiers d'en-tête

> ⚠ Rappel : vous ne devez modifier que le fichier `tp.c`. Ce sera le seul pris en compte pour la notation.

Les autres fichiers fournis dans l'archive `tp.tgz` ne doivent **pas** être modifiés, mais trois d'entre eux contiennent des fonctions utiles que vous pouvez utiliser. Elles sont décrites dans les fichiers d'en-tête (`.h`) suivants.

1. Le fichier le plus important est `number.h`. Il contient la définition du type `number`. Ce type est une structure qui permet de représenter de très grands entiers positifs ou nuls. Un entier est codé par un tableau de chiffres `digit` dont la taille `n` est donnée. Les nombres sont codés dans une base entre 2 et 10, qui est une variable globale `BASE`.

   À l'indice 0 de ce tableau de chiffres (c'est-à-dire en `digit[0]`) se trouve le chiffre des unités du nombre représenté. Par exemple, le tableau de 4 chiffres `{5,3,0,1}`, quand `BASE` vaut 10, représente l'entier :

$$x = 5 \cdot 10^0 + 3 \cdot 10^1 + 0 \cdot 10^2 + 1 \cdot 10^3 = 1035.$$

   > ⚠ Attention : à cause des zéros inutiles, un entier a plusieurs représentations possibles. Ainsi, pour l'entier x ci-dessus, le tableau `digit` peut avoir les `n` $= 7$ cases suivantes : `{5,3,0,1,0,0,0}`. En effet, dans toute base, les zéros de tête dans son écriture dans la base (qui se trouvent donc en fin de tableau `digit`) ne changent pas la valeur du nombre. Par exemple, 1035 et 0001035 représentent bien le même entier.

   Le fichier `number.h` contient aussi des fonctions permettant des manipulations de base sur le type `number` (comme allouer un nouvel entier, désallouer un ou plusieurs entiers, convertir un entier en chaîne de caractères et inversement, et enfin, générer des entiers aléatoires).
2. Le fichier `alloc.h` décrit des fonctions simplifiant l'allocation dynamique.
3. Le fichier `tools.h` contient des fonctions calculant le maximum et le minimum de deux entiers, ainsi qu'une macro `SWAP` permettant d'échanger deux variables.

2.1.2 Le fichier `Makefile`

Le fichier `Makefile` fourni ne doit pas être modifié, car votre fichier `tp.c` sera testé avec ce `Makefile` (et avec les autres fichiers fournis). Il est volontairement exigeant : par défaut, les *warnings* sont traités comme des erreurs car vous devez fournir un source `tp.c` qui compile sans aucun avertissement. Le `Makefile` construit l'exécutable `tp`.

Les possibilités sont :

- Par défaut, de construire l'exécutable `tp` en traitant les warnings comme des erreurs (l'option `-Werror` impose d'éviter tout warning) et avec des options `-O0 -g` permettant d'utiliser `gdb` et `valgrind`.

```
$ make all      # ou bien
$ make
```

- De construire l'exécutable `tp` sans l'option `-Werror` et en mode optimisé `-O3`, sans utilisation possible de `gdb`.

```
$ make nowerror
```

- De nettoyer :

```
$ make clean
```

  Le nettoyage supprime les fichiers générés (non seulement les fichiers `.o` mais aussi les fichiers `.d`, utilisés pour gérer les dépendances dans le `Makefile`).

3 Interface pour tester votre programme

Une fois le programme compilé, vous pouvez le tester selon deux modes :

- Mode calculatrice, ou
- Mode tests aléatoires.

La commande :

```
$ ./tp --help
```

fournit une aide des options disponibles. Voir la section 5 pour des explications supplémentaires et des exemples.

4 Fonctions à écrire

Les descriptions des fonctions à écrire sont détaillées dans le fichier `tp.c` (et non dans le `.h`, pour ce fichier). Dans les questions suivantes, on identifie les variables `X`, `Y` avec les entiers qu'ils représentent.

Exercice 1 – Nombre de chiffres significatifs

Écrivez la fonction `int number_length(number *X)` qui renvoie le nombre de chiffres significatifs de `X`. Par exemple si `X->digit` est le tableau `{0,1,2,8,0,0}`, la fonction doit renvoyer 4, parce que `X` code l'entier 8210, qui s'écrit avec 4 chiffres (sans les 0 de tête).

Vous pouvez tester cette fonction uniquement sous l'interface, en utilisant `len`.

Exercice 2 – Comparaison de deux nombres en précision arbitraire

Écrivez la fonction `int number_sign(number *X, number *Y)` qui renvoie `-1` si l'entier `X` est strictement plus petit que l'entier `Y`, `1` si `X` est strictement plus grand que `Y`, et `0` si `X` et `Y` sont égaux.

Vous pouvez tester cette fonction avec les opérateurs `==`, `<` et `>` de l'interface, ainsi qu'avec l'option `--comp`.

Exercice 3 – Addition

Écrivez la fonction `number *number_addition(number *X, number *Y)` qui alloue un nouveau nombre qui représente `X+Y`, et renvoie un pointeur sur lui. L'algorithme d'addition en colonne est brièvement rappelé en section 6. Notez que si `X` a n_X chiffres et `Y` a n_Y chiffres, le nombre de chiffres du résultat est au maximum $1 + \max(n_X, n_Y)$. Vous pouvez laisser des zéros inutiles dans le résultat.

Vous pouvez tester cette fonction avec l'opérateur `+` de l'interface, ainsi qu'avec l'option `--add`.

Exercice 4 – Soustraction

Écrivez la fonction `number *number_substraction(number *X, number *Y)` qui alloue un nouveau nombre qui représente `|X-Y|`, et renvoie un pointeur sur lui. Autrement dit, la fonction doit soustraire le plus petit nombre au plus grand. L'algorithme est aussi rappelé en section 6. Notez que si `X` a n_X chiffres et `Y` a n_Y chiffres, le nombre de chiffres du résultat est au maximum $\max(n_X, n_Y)$. À nouveau, vous pouvez laisser des zéros inutiles dans le résultat.

Vous pouvez tester cette fonction avec l'opérateur `-` de l'interface, ainsi qu'avec l'option `--sub`.

Les questions suivantes sont utiles pour implémenter les deux versions de la multiplication.

Exercice 5 – Décalage d'un nombre en ajoutant des 0 en tête de la représentation

Écrivez la fonction `number *shift(number *X, int k)` qui alloue et renvoie le nombre obtenu en multipliant `X` par $\texttt{BASE}^{\texttt{k}}$ (où `k` est un entier positif ou nul), c'est-à-dire, en ajoutant `k` zéros en tête du tableau `X->digit`.

Par exemple, si `X->digit` vaut `{1,0,8,0}` et `k` vaut `3`, la fonction doit allouer un `number` dont le tableau `digit` vaudra `{0,0,0,1,0,8,0}`, et renvoyer un pointeur sur lui.

Exercice 6 – Chiffres de poids faible d'un entier

Écrivez la fonction `number *low(number *X, int m)` qui alloue et renvoie le nombre obtenu en ne gardant que les `m` premiers chiffres du tableau `X->digit` (où `m` est un entier positif ou nul).

Par exemple si `X->digit` vaut `{5,2,1,0,8,0}` et `m` vaut 2, la fonction renvoie un pointeur sur un nombre dont le tableau `digit` vaut `{5,2}`.

Exercice 7 – Chiffres de poids fort d'un entier

Écrivez la fonction `number *high(number *X, int m)` qui alloue et renvoie le nombre obtenu en retirant les `m` premiers chiffres du tableau `X->digit` (où `m` est un entier positif ou nul).

Par exemple si `X->digit` vaut `{5,2,1,0,8,0}` et `m` vaut 2, la fonction renvoie un pointeur sur un nombre dont le tableau `digit` vaut `{1,0,8,0}`.

Exercice 8 – Mise à la même taille de deux entiers

Écrivez la fonction `void align_numbers(number *X, number *Y)` qui ajoute si besoin des zéros inutiles à l'un des deux nombres représentés par `X` et `Y` pour que les tableaux `X->digit` et `Y->digit` soient de même taille. Après l'appel, `X->digit` et `Y->digit` ont donc la même taille, qui est le maximum des tailles d'origine, et la valeur des entiers représentés par `*X` et `*Y` n'est pas changée.

Attention à bien mettre à jour les **deux** champs du nombre modifié. Notez aussi que contrairement aux autres fonctions, celle-ci modifie le `number` pointé par un de ses arguments (en réallouant le tableau `digit`, notamment), mais elle n'alloue pas de nouvelle structure `number`.

Par exemple, si `X->digit` vaut `{1,2,3,4}` et `Y->digit` vaut `{1,2,3,4,5,6,7,8}`, alors après appel à `align_numbers(X, Y)`, le tableau `X->digit` vaudra `{1,2,3,4,0,0,0,0}` et `Y->digit` sera inchangé. Par ailleurs, `X->n` vaudra 8 (au lieu de 4 initialement) et `Y->n` vaudra 8 (il est inchangé).

Exercice 9 – Multiplication selon l'algorithme récursif naïf

Écrivez la fonction `number *number_multiplication_recursive(number *X, number *Y)` qui alloue un nouveau nombre représentant `X`×`Y`, et renvoie un pointeur sur lui, en utilisant l'algorithme de multiplication récursive rappelé en section 6. Notez que si `X` et `Y` ont n chiffres, le nombre de chiffres du résultat est au maximum $2n$. Vous pouvez laisser des zéros inutiles dans le résultat.

Vous pouvez tester cette fonction avec l'opérateur `*` de l'interface, ainsi qu'avec l'option `--rec`.

Exercice 10 – Multiplication selon l'algorithme de Karatsuba

Écrivez la fonction `number *number_multiplication_karatsuba(number *X, number *Y)` qui alloue un nouveau nombre représentant `X`×`Y`, et renvoie un pointeur sur lui, en utilisant l'algorithme de multiplication de Karatsuba, rappelé en section 6. Notez que si `X` et `Y` ont n chiffres, le nombre de chiffres du résultat est au maximum $2n$. Vous pouvez laisser des zéros inutiles dans le résultat.

Vous pouvez tester cette fonction avec l'opérateur `*k` (ou `**`) de l'interface, ainsi qu'avec l'option `--kar`.

5 Interface de test : exemples

5.1 Mode calculatrice

Ce mode est interactif : tapez une expression arithmétique usuelle contenant des opérations, elle seront évaluées et l'interface précisera si le calcul est correct ou non. Les opérations arithmétiques se notent comme suit : `+` (addition), `-` (soustraction), `*` (multiplication récursive naïve) et `**` ou `*k` (multiplication « Karatsuba »).

En plus de ces opérations, vous pouvez changer de base par la commande `base`, calculer la longueur d'un nombre par la commande `len`, et comparer des entiers par `==`, `<` ou `>`. À part pour la commande `base`, l'interface ne fait que lire des entiers, les convertir dans le type `number`, et appeler les fonctions que vous avez écrites.

Une seule option est prise en compte en mode interactif : l'option `--base=<entier>`. Elle permet de fixer la base de départ. Vous pouvez aussi la changer ensuite avec la commande `base` de l'interface. Les nombres sont saisis comme on en a l'habitude : chiffre des unités à droite. Ils sont traduits en `number` par l'interface (en particulier, les chiffres sont inversés dans le tableau `digit`), et les fonctions que vous avez écrites sont ensuite appelées.

Les deux sessions qui suivent montrent :

- à gauche, ce que vous devriez obtenir en compilant et exécutant le programme sans avoir codé les fonctions,
- à droite, ce que vous devriez obtenir après avoir codé correctement les fonctions demandées.

Lorsqu'une ligne de calcul contient plusieurs opérations, comme $123 - 123 - 1$, le détail de chaque opération est réalisé (ici, $123 - 123$, puis, ce résultat moins 1). Les squelettes des fonctions que nous vous fournissons calculent toutes la valeur 0, au lieu de faire l'opération demandée. C'est donc par hasard que dans l'exemple ci-dessous à gauche, la réponse intermédiaire `123 - 123` est correcte. Enfin, la fonction `len` appelle la fonction `number_length` demandée à l'exercice 1. Sur l'exemple à gauche, elle aurait dû renvoyer 3.

Session sous l'interface initialement

```
$ ./tp --base=2
Mode calculatrice. Calculs en base 2.
>>> 111 + 1
✖ (attendu 1000, calculé 0)
-- Calculé : 0
>>> base(10)
Nouvelle base : 10
>>> 123 - 123 - 1
✔ 123 - 123 = 0
✖ (attendu : -1, calculé 0)
-- Calculé : 0
>>> len(0123)
0
>>> exit
Bye!
```

Session sous l'interface, fonctions correctes

```
$ ./tp --base=2
Mode calculatrice. Calculs en base 2.
>>> 111 + 1
✔ 111 + 1 = 1000
-- Calculé : 1000
>>> base(10)
Nouvelle base : 10
>>> 123 - 124
✔  123 - 124 = -1
-- Calculé : -1
>>> len(0123)
3
>>> exit
Bye!
```

5.2 Mode tests aléatoires

Il est possible de générer des tests aléatoires avec une commande du type :

```
$ ./tp --add --num=10 --min=200 --max=10000
```

La première option, `--add`, demande le lancement de tests pour l'addition. En mode « tests aléatoires », l'une exactement des options `--comp` `--add`, `--sub`, `--rec` ou `--kar` doit être donnée. Elle lance les tests pour la comparaison de deux entiers, l'addition, la soustraction, la multiplication récursive ou la multiplication Karatsuba.

Le nombre de tests peut être ajusté par l'option `--num` (10 par défaut). Le nombre de chiffres des entiers utilisés dans les tests est compris entre les nombres donnés par `--min` (3 par défaut) et `--max` (10 par défaut). Voir `./tp --help` pour plus de détails.

6 Annexe : rappel d'algorithmes pour les opérations arithmétiques

6.1 Addition

L'algorithme d'addition que vous devez implémenter est l'algorithme standard d'addition en colonne. On additionne d'abord les chiffres des unités des deux nombres, puis ceux des dizaines (en base 10), etc. Les premiers chiffres à additionner sont donc ceux en position `0` dans le tableau `digit`. Si l'une des additions de chiffres produit un résultat supérieur ou égal à la base, on propage une retenue. Un exemple est donné ci-dessous.

Exemple pour l'algorithme d'addition

Dans l'exemple ci-contre, on commence par « aligner » les unités des entiers à additionner (4518 et 95725). La première addition, $8+5=13$, propage une retenue. Notez qu'il est possible qu'un des nombres soit plus court que l'autre (c'est le cas sur l'exemple). Par ailleurs, une retenue peut se propager à gauche du dernier chiffre des nombres.

$$\begin{array}{rr} & {\scriptstyle 1\ 1\ \ \ 1} \\ & 4\,5\,1\,8 \\ + & 9\,5\,7\,2\,5 \\ \hline & 1\,0\,0\,2\,4\,3 \end{array}$$

6.2 Soustraction

L'algorithme d'addition que vous devez implémenter est l'algorithme standard de soustraction en colonne. On soustrait le plus petit entier au plus grand, pour obtenir un résultat positif ou nul. On soustrait d'abord les chiffres des unités des deux nombres, puis ceux des dizaines (en base 10), etc. Si une des soustraction produit un résultat strictement négatif, on ajoute la base au résultat pour obtenir le chiffre à reporter, et on propage une retenue au niveau du nombre que l'on soustrait. Un exemple est donné ci-dessous.

Exemple pour l'algorithme de soustraction

Dans l'exemple ci-contre, on commence par « aligner » les unités des entiers à soustraire (95725 et 4518). La première soustraction, $5-8=-3$ produit un nombre négatif. On lui ajoute la base (10) pour reporter le chiffre $-3+10=7$, et on propage une retenue. La soustraction suivante sera donc $2-(1+1)=0$.

$$\begin{array}{rr} & 9\,5\,7\,2\,{\scriptstyle 1}5 \\ - & 4\,5\,{\scriptstyle 1}1\,8 \\ \hline & 9\,1\,2\,0\,7 \end{array}$$

6.3 Multiplication naïve

Algorithme récursif de multiplication

Entrée : x, y deux entiers naturels, x de n_x chiffres et y de n_y chiffres, écrits en base B.
Sortie : $x \times y$ sur $2n$ chiffres, où $n = \max(n_x, n_y)$.

1. Si x et y n'ont pas le même nombre de chiffres, ajouter des 0 inutiles à celui ayant le moins de chiffres pour que les deux entiers soient représentés sur n chiffres, où $n = \max(n_x, n_y)$.
   Soient $x_{n-1} \cdots x_0$ et $y_{n-1} \cdots y_0$ les écritures de x et y en base B (x_0 et y_0 sont les chiffres des unités).
2. Si $n = 1$, renvoyer $x_0 \times y_0$ (⚠ le résultat peut avoir 2 chiffres : $5 \times 5 = 25$ donnera le tableau `{5,2}`).
3. Sinon, soit $m = \lceil n/2 \rceil$. Poser :
   $$x^+ = x_{n-1} \cdots x_m, \quad \text{et} \quad x^- = x_{m-1} \cdots x_0,$$
   $$y^+ = y_{n-1} \cdots y_m, \quad \text{et} \quad y^- = y_{m-1} \cdots y_0.$$
4. Calculer :
   $$a = \texttt{Multiplication}(x^+, y^+),$$
   $$b = \texttt{Multiplication}(x^-, y^+),$$
   $$c = \texttt{Multiplication}(x^+, y^-),$$
   $$d = \texttt{Multiplication}(x^-, y^-).$$
5. Renvoyer $a \cdot B^{2m} + (b+c) \cdot B^m + d$. Notez que la « multiplication » par B^k est juste un décalage.

6.4 Multiplication par l'algorithme de Karatsuba

Pour l'algorithme de Karatsuba, vous devez implémenter la version ci-dessous.

> ⚠ Attention, cet algorithme n'est pas tout à fait celui du cours !

Algorithme de Karatsuba

Entrée : x, y deux entiers naturels, x de n_x chiffres et y de n_y chiffres, écrits en base B.
Sortie : $x \times y$ sur $2n$ chiffres, où $n = \max(n_x, n_y)$.

1. Si x et y n'ont pas le même nombre de chiffres, ajouter des 0 inutiles à celui ayant le moins de chiffres pour que les deux entiers soient représentés sur n chiffres, où $n = \max(n_x, n_y)$.
   Soient $x_{n-1} \cdots x_0$ et $y_{n-1} \cdots y_0$ les écritures de x et y en base B (x_0 et y_0 sont les chiffres des unités).
2. Si $n = 1$, calculer et renvoyer $x_0 \times y_0$.
3. Sinon, soit $m = \lceil n/2 \rceil$. Poser :
   $$x^+ = x_{n-1} \cdots x_m \quad \text{et} \quad x^- = x_{m-1} \cdots x_0,$$
   $$y^+ = y_{n-1} \cdots y_m \quad \text{et} \quad y^- = y_{m-1} \cdots y_0.$$
4. Calculer :
   $$a = \texttt{Karatsuba}(x^+, y^+),$$
   $$c = \texttt{Karatsuba}(x^-, y^-),$$
5. Calculer :
   $$d_x = |x^+ - x^-| \text{ et } s_x = \text{signe}(x^+ - x^-),$$
   $$d_y = |y^+ - y^-| \text{ et } s_y = \text{signe}(y^+ - y^-).$$
   $$d = \texttt{Karatsuba}(d_x, d_y).$$
   $$b = \begin{cases} a + c - d & \text{si } s_x = s_y, \\ a + c + d & \text{si } s_x \neq s_y. \end{cases}$$
6. Renvoyer $a \cdot B^{2m} + b \cdot B^m + c$.