Langage C - projet

1 Modalités

Le projet doit être réalisé en binôme. Les soutenances auront lieu au mois de mai (après le 9 mai), la date exacte sera communiquée ultérieurement. Pendant la soutenance, les membres de binôme devront chacun montrer leur maîtrise de la totalité du code.
Le rendu sera à déposer sur moodle.

2 Présentation de projet

Le but du projet est d'écrire une bibliothèque de fonctions qui implémentent des opérations sur les entiers avec un nombre arbitraire de chiffres.

3 Type de données

Les entiers seront représentés par des listes doublements chaînées. La structure suivante définit un élément de la liste :

```
typedef struct chiffre{
  struct chiffre  *suivant;
  char c;
  struct chiffre *precedent;
  } chiffre;
```

La structure sert à stocker un chiffre représenté par le caractère correspondant : '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'.
Les champs `precedent` et `suivant` pointent respectivement vers le bloc précédent et suivant.
On utilisera la structure

```
typedef struct{
    char signe;     /* soit '+' soit '-'  */
    size_t len;     /* longueur de la liste */
    chiffre *premier; /* pointeur vers le premier élément de la liste*/
    chiffre *dernier; /*pointeur vers le dernier élément de la liste*/
} unbounded_int;
```

pour stocker

- les pointeurs vers le premier et le dernier bloc de la liste de `chiffre`,
- le signe stocké comme un caractère, soit '+' soit '-',
- `len` - le nombre de chiffres sur la liste.

La figure 1 montre la représentation d'un nombre à 6 chiffres.

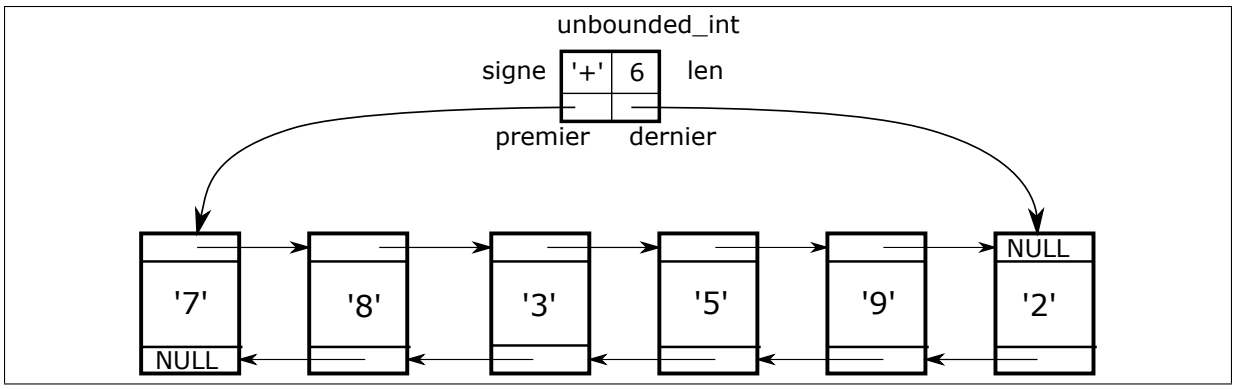

FIGURE 1 – La représentation du nombre 783592. La structure `unbounded_int` stocke les pointeurs vers le premier et dernier élément de la liste de chiffres. Le champ `signe` de `unbounded_int` contient le signe, ici `'+'`, du nombre. Le champ `len` contient la longueur 6 de la liste. La représentation du nombre −783592 est identique, sauf pour le champ `signe` qui stockera le caractère `'-'`. Notez que la liste de `chiffres` n'est pas cyclique, le champ `precedent` du premier élément et le champ `suivant` du dernier sont `NULL`.

Le nombre 0 aura un seul est représenté par un seul caractères `'0'`. Le signe de 0 peut être aussi bien `'+'` que `'-'`.

4 Travail demandé

Le travail demandé est divisé en deux parties décrites ci-dessous. Dans la première, vous aurez à écrire et tester des fonctions permettant de construire des objets `unbounded_int`, ou implémentant des opérations arithmétiques sur les entiers `unbounded_int`.

Dans la seconde partie, vous aurez à écrire un mini-interpréteur permettant de lire un fichier dont le nom sera un paramètre de `main`. Ce fichier contiendra des instructions décrivant des opérations à effectuer sur des objets `unbounded_int`. Les résultats de ces opérations seront écrits dans un autre fichier texte dont le nom sera un autre paramètre de `main`.

4.1 Fonctions sur les `unbounded_int`

Certains de fonctions suivantes retournent la structure `unbounded_int`. Si les appels à `malloc()` échouent pour indiquer le problème la fonction doit retourner un `unbounded_int` avec le champ `signe == '*'`.
Écrire les fonctions suivantes :

(1) `unbounded_int string2unbounded_int(const char *e)` prenant en argument l'adresse d'une chaîne de caractères représentant un entier (*e.g.* `"421"`) et renvoyant une valeur `unbounded_int` représentant cet entier suivant le format présenté à la section 3. Si la chaîne pointée par `e` ne représente pas un entier la fonction retournera un `unbounded_int` avec le champ `signe == '*'`.

(2) `unbounded_int ll2unbounded_int(long long i)` qui prend en argument un nombre `long long` et retourne la structure `unbounded_int` représentant ce nombre.

(3) `char *unbounded_int2string(unbounded_int i)` qui prend en argument une structure `unbounded_int` représentant un entier et retourne cet entier sous forme d'une chaîne de caractères.
Cette fonction est l'inverse de la fonction `string2unbounded_in()`, par exemple `unbounded_int2string(string2unbounded_in("-4543676543298" ))` doit retourner un pointeur vers la chaîne de caractères `"-4543676543298"`.

(4) `int unbounded_int_cmp_unbounded_int(unbounded_int a, unbounded_int b)` qui retourne une des valeurs $-1, 0, 1$ si, respectivement, $a < b$, $a == b$, $a > b$ (ou a, b sont les entiers représentés par `a` et `b`).

(5) `int unbounded_int_cmp_ll(unbounded_int a, long long b)` renvoyant la même chose que la fonction précédente, mais où le second argument est de type `long long`.

(6) `unbounded_int unbounded_int_somme(unbounded_int a, unbounded_int b)` renvoyant la représentation de la somme de deux entiers représentés par `a` et `b`,

(7) `unbounded_int unbounded_int_difference( unbounded_int a, unbounded_int b)` renvoyant la représentation de leur différence,

(8) `unbounded_int unbounded_int_produit( unbounded_int a, unbounded_int b)` renvoyant la représentation de leur produit,

Vous devrez écrire un programme qui teste toutes ces fonctions.

4.2 Interpréteur

La deuxième partie du projet consiste à écrire un mini-interpréteur `calc_unbounded_int`.
Ce programme sera exécuté en ligne de commande de façon suivante :

```
./calc_unbounded_int -i source.txt -o dest.txt
```

L'option `-i` du programme donne le nom de fichier texte qui contient une suite d'instructions à exécuter par l'interprète. Dans l'option `-o` on passe le nom de fichier où `calc_unbounded_int` écrira les résultats.
On suppose que les deux options sont facultatives : si l'option `-i` est absente `calc_unbounded_int` lira les instructions depuis le flot d'entrée standard `stdin` et si l'option `-o` est absente le programme écrira les résultats dans le flot `stdout`.
Le fichier lu par l'interpréteur contient trois types d'instructions, réparties sur des lignes distinctes :

(A) `variable = entier_ou_variable op entier_ou_variable`
`entier_ou_variable` est soit un entier soit une variable. `op` est une de trois opérations : `*, +, -`. L'interpréteur doit effectuer l'opération, le résultat de l'opération devient la valeur de la `variable`.

(B) `variable = entier`
`entier` devient la nouvelle valeur de la `variable`.

(C) `print variable`
L'interpréteur écrira dans le fichier de sortie une ligne sous forme
`variable = valeur`
où `valeur` est la valeur de `variable`.

On supposera que la valeur initiale de chaque variable est 0, en particulier `print a` sur une variable `a` non-initialisée écrira `a = 0` dans le fichier de sortie.
Dans un premier temps, pour simplifier, on supposera que les noms de variables sont composés d'une seule lettre minuscule (il y a donc un nombre fini de variables : `a, b, c, ... ,z`).

Exemple. Voici un exemple de fichier texte qui pourrait servir comme le fichier d'entrée pour l'interpréteur :

```
a =897676764344676545687876565
a = a * a
a = a - 7867656443555498966543
b = 98987676565
   b = 3 * b
a=a - b
print a
print b
c=    a  -  -565543443423
print c
```

On suppose que
- chaque ligne contient au plus une instruction,
- il est possible d'avoir des lignes vides (sans instruction),
- il n'y a pas de parenthèses,
- il y a des espaces (au moins un de chaque côté) autour de chaque opération `* , + , -`,
- les entiers positif peuvent être écrits soit comme `23432` soit comme `+98676565` avec `+` au début suivi immédiatement d'un chiffre,
- les nombres négatifs s'écrivent comme `-657867654` avec le signe `-` suivi immédiatement d'un chiffre,
- des espaces à gauche ou à droite de `=` sont autorisés mais pas obligatoires,
- on peut avoir zéro ou plusieurs espaces au début de chaque ligne.

Pour la lecture du fichier on préfère que vous n'imposiez aucune contrainte sur la longueur de la ligne. Si vous ne savez pas le faire alors `calc_unbounded_int` doit être capable de lire au moins les lignes de longueur 1024.

4.3 Fonctions auxiliaires

Aussi bien dans la première que dans la deuxième partie du projet vous pouvez écrire les fonctions auxiliaires de votre choix. Toutes les fonctions auxiliaires doivent être déclarées avec l'attribut `static`.

5 Indications sur les algorithmes

Pour implémenter l'addition et soustraction il est commode de commencer par écrire les fonctions auxiliaire pour
- additionner deux `unbounded_int` non-négatifs,
- implementer la soustraction `a - b` pour `a` et `b` positifs et tels que `a >= b`.

Une fois ces deux opérations implémentées il est facile d'implémenter l'addition et la soustraction dans le cas général en utilisant le changement de signe :

$$a+b=\begin{cases} a+b & \text{si } a,b\geqslant 0\\ -(|a|+|b|) & \text{si } a,b\leqslant 0\\ a-|b| & \text{si } a\geqslant 0, b<0\\ b-|a| & \text{si } b\geqslant 0, a<0 \end{cases} \tag{1}$$

et

$$a-b=\begin{cases} a-b & \text{si } a,b\geqslant 0\\ |b|-|a| & \text{si } a,b\leqslant 0\\ a+|b| & \text{si } a\geqslant 0, b<0\\ -(b+|a|) & \text{si } b\geqslant 0, a<0 \end{cases} \tag{2}$$

5.1 Transformation de `char` en nombre

Rappelons que si `c` est l'un des caractères `'0','1','2','3','4','5','6','7','8','9'` alors `c - '0'` est le chiffre correspondant.
Par exemple `'7' - '0'` vaut 7.

Dans le sens inverse, si i est un entier entre 0 et 9 alors `(char)( i + '0')` est le caractère correspondant, par exemple `(char) (8 + '0')` vaut `'8'`.

5.2 Addition

Vous devez implementer l'algorithme d'addition que l'on utilise lorsqu'on fait l'addition à la main sur papier.
Soient a et b deux entier positifs. Si $a=a_na_{n-1}\dots a_1a_0$ et $b=b_ma_{m-1}\dots b_1b_0$ où a_i et b_i sont le i-ème chiffres de a et de b et $c=a+b$ alors i-ème chiffre de c, c_i, s'obtient comme

$$c_i=(a_i+b_i+r)\%10 \tag{3}$$

où r est la retenue (`% 10` pour la notation à la C pour `modulo 10`).
La retenue pour le calcul de chiffre suivant c_{i+1} est

$$r=(a_i+b_i+r)/10 \tag{4}$$

(la division entière).

5.3 Soustraction

Supposons que nous voulons calculer $a-b$ avec $a>b>0$.
Soit $a=a_na_{n-1}\dots a_1a_0$ les chiffres de a, $b=b_mb_{m-1}\dots b_1b_0$ les chiffres de b. Puisque $a>b$ nous avons $n\geqslant m$.
Il faut implémenter l'algorithme classique que nous utilisons lorsqu'on fait l'opération sur papier.
Initialement la retenue r_0 est 0.
Pour obtenir le chiffre c_i du résultat

$$c_i = \begin{cases} a_i - b_i + r_i & \text{si } a_i - b_i + r_i \geqslant 0 \\ a_i - b_i + r_i + 10 & \text{si } a_i - b_i + r_i < 0 \end{cases} \tag{5}$$

et la retenue suivante r_{i+1} :

$$r_{i+1} = \begin{cases} 0 & \text{si } a_i - b_i + r_i \geqslant 0 \\ -1 & \text{si } a_i - b_i + r_i < 0 \end{cases} \tag{6}$$

5.4 Multiplication

Soit $a = a_{n-1}a_{n-2}\dots a_1a_0$ les chiffres de a, $b = b_{m-1}b_{m-2}\dots b_1b_0$ les chiffres de b.
L'algorithme utilisé pour le calcul du produit $a * b$ sur le papier consiste à multiplier a successivement par b_0, ensuite b_1, b_2 etc. et après le décalage approprié faire la somme.
Nous pouvons faire la multiplication par un chiffre et l'addition en même temps.
Le pseudocode est le suivant :
On initialise tous les chiffres c_i de résultat $c = c_{m+n-1}\dots c_0$ avec la valeur 0. Ensuite on lance l'algorithme suivant :

```
for( j = 0; i < m; i++){ /* boucle sur les chiffres de b*/
  r = 0;
  if( b[j] == 0 )
     continue;
  for(i=0; i < n; i++ ){ /* boucle sur les chiffres de a*/
    int v =  c[i+j] + a[i]*b[j] + r;
    c[i+j]  = v % 10;
    r = v / 10;
  }
  c[j + n] = r;
}
```

Dans cet algorithme `c[i+j]` est le $(i+j)$ème chiffre de `c` , `a[i]` $i$ème chiffre de `a` et `b[j]` le $j$ème chiffre de `b`. Dans ce pseudo-code on assume que les chiffres sont des nombres entre 0 et 9 et non pas de caractères `'0'`, ..., `'9'` .

5.5 Quotient

Calculer a/b avec les nombres écrits en base de 10 ne semble pas facile. Quand on le fait sur le papier, on « dévine » les chiffres.
Pour cette raison, on ne demande pas d'implémenter cette opération.
Notons que si a et b sont écrits en binaire le calcul de a/b et de $a\%b$ est très simple.
Une solution viable serait peut-être de transformer a et b en binaire, de calculer le quotient a/b et le reste $a\%b$ en binaire et retraduire les résultats en base 10.
L'implémentation des opérations de quotient et de modulo `%` n'est donc pas obligatoire mais le faire vous permettra d'obtenir des points supplémentaires (sauf si vous avez déjà 20 points !).

6 Extensions

Bien réalisé, le projet de base présenté ci-dessus permet d'obtenir une bonne note. Mais si vous visez une très bonne note il faut implémenter l'interpréteur qui admet les variables dont le nom peut être composé de plusieurs lettres, sans aucune restriction sur la longueur de nom. Bien sûr le nombre de variables ne doit pas être limité non plus.

7 Organisation du code

Vous créez un répertoire `C_XXXX_YYYY` où XXXX et YYYY sont les noms de membres de binôme.
Ce répertoire doit contenir :

- le fichier `README` avec les noms et numéro étudiants de binôme,
- le `Makefile`,
- le fichier `unbounded_int.c` avec toutes les fonctions de la section 4.1 et le fichier `unbounded_int.h` correspondant,
- le fichier `test_unbounded.c` qui contient le code du programme qui teste les fonctions définies dans `unbounded_int.c`,
- le fichier `calc_unbounded_int.c` avec le code de l'interpréteur.

Les fichiers source doivent être compilés avec les options `-Wall -g -pedantic`.
`make clean` doit supprimer les exécutables et les fichiers `*.o` de telle sort que `make` qui suivra recompile tous les fichiers.
Vous déposerez le répertoire `C_XXXX_YYYY` uniquement avec les fichier indiqués ci-dessus sur moodle (vous pouvez compresser le répertoire soit comme `C_XXXX_YYYY.zip` soit `C_XXXX_YYYY.tar.gz`).

Vous n'avez pas le droit de donner d'autres noms à vos fichiers. Vous programmes seront examinés par un détecteur de plagiat. Si vous donnez d'autres noms à vos fichier cela rend la détection de plagiat plus fastidieuse (mais pas impossible) et le surcoût de travail que cela engendre pour nous sera pénalisé par des points négatifs dans la note finale du projet.