IFT2035 – Travail pratique #1 – 2016.09.23

GESTION MÉMOIRE ET POINTEURS

Marc Feeley

Le TP1 a pour but de vous faire pratiquer la programmation impérative et en particulier les concepts suivants : les énoncés de contrôle, la gestion mémoire manuelle, et les pointeurs.
Vous avez à réaliser un programme en C ainsi que rédiger un rapport contenant une analyse du programme.

---

1 Programmation

Vous devez réaliser l'application suivante dans le langage C en exploitant les forces de ce langage pour exprimer au mieux votre programme (e.g. l'arithmétique sur les pointeurs si c'est approprié).

L'application est une petite calculatrice à précision illimitée. Il faut représenter des nombres entiers avec des listes de chiffres, de telle façon que la calculatrice supporte des nombres de longueur arbitraire. Le programme doit implanter les opérateurs +, -, * et ? (expliqué ci-dessous) ainsi que des variables représentées par une lettre minuscule a..z. Notez que l'opérateur de division n'existe pas dans la syntaxe des expressions.

Chaque expression est sur une ligne et la calculatrice attends l'entrée en imprimant "`>`" en guise d'incitateur. Après la fin de ligne le programme imprime le résultat sur une nouvelle ligne. Les expressions sont sous forme postfixe, donc une interaction valide avec le programme pourrait être :

```
> 10 15 + 2 *
50
```

Pour l'affectation d'une variable, il faut utiliser la syntaxe : ⟨*expression*⟩ `=`⟨*variable*⟩. Par exemple pour assigner 100 à la variable "a" :

```
> 100 =a
100
> a 1 +
101
```

Notez qu'il n'y a pas d'espace entre le signe = et le nom de la variable. Le résultat de l'affectation est la valeur affectée. Après l'affectation on peut utiliser la variable dans des expressions. Les noms de variable valides n'ont qu'un caractère. C'est une erreur d'utiliser une variable qui n'a pas encore reçu de valeur. Les nombres négatifs ne sont pas permis dans les expressions, mais le résultat d'un calcul peut être négatif.

Un exemple plus long d'utilisation est :

```
> 1 1 + 1 - 48 *
48
> 1000 1000 * =a
1000000
> 1 a -
-999999
> a 1 + =a
```

```
1000001
> a 1 * =b
1000001
> 1001 =b
1001
> b 100 + =b
1101
> b b b b * * *
1469431264401
> 1000 =b 1000 * =c
1000000
```

Au fur et à mesure que le programme exécute, il faut libérer la mémoire qui ne sera plus utilisée. Il faut faire attention de ne pas libérer la mémoire des variables qui pourraient être utilisées plus tard.

Le programme doit lire des lignes et imprimer les réponses jusqu'à ce qu'il trouve un EOF (End Of File ou `^D`). Il ne doit pas y avoir de limite sur la longueur des lignes et la taille des nombres. À ce moment, le programme finit en libérant toute la mémoire allouée qui n'a pas encore été libérée.

Une première version des structures nécessaires pour la représentation des nombres à précision infinie pourrait être :

```
struct num { int negatif; struct cell *chiffres; };
struct cell { char chiffre; struct cell *suivant; };
```

Pour faciliter l'implantation, un chiffre sera un caractère de "0" à "9". Les nombres sont représentés par une liste de caractères qui sont les chiffres constituant le nombre. Les chiffres dans la liste sont du chiffre de poids faible au chiffre de poids élevé. De plus, il ne doit pas y avoir de "0" dans le chiffre de poids le plus élevé. Le nombre entier 0 est représenté par une liste vide de chiffres, c'est-à-dire NULL (cela simplifie les algorithmes). Pour faire les calculs, il suffit d'utiliser des algorithmes "naïfs" d'addition, soustraction et multiplication. Les mêmes algorithmes qu'on utilise sur papier marchent (même s'il y a des algorithmes beaucoup plus efficaces et complexes).

La structure `num` est une représentation signe et magnitude, donc le champ "negatif" est vrai (valeur différente de 0) si le nombre est négatif.

Vous ne devez pas copier les `num` car cela pourrait gaspiller de la mémoire lorsque les nombres sont grands. Vous devez modifier le type `num` pour ajouter un compteur de référence que votre code devra maintenir à jour explicitement. L'opérateur unaire postfixe "?" permet d'extraire le compteur de référence d'un nombre, c'est-à-dire qu'une expression suivie de l'opérateur "?" transforme le nombre entier de l'expression en un nombre entier égal au compteur de référence.

```
> 5 10 * =x =y
50
> 2 25 * =z
50
> x ?
2
> z ?
1
```

Le nombre `50` calculé par "`5 10 *`" n'est stocké qu'une fois dans la mémoire. Puisqu'il est référé par la variable x et y son compteur de référence est égal à 2 après l'exécution de l'expression "`5 10 * =x =y`". Si ensuite on fait :

```
> 42 =y
42
> 11 =x
11
```

À ce moment il ne reste pas des références au nombre `50` calculé par "`5 10 *`" donc la mémoire occupée par ce nombre doit être récupérée.

Votre programme doit être robuste. S'il y a un problème, le programme doit donner un message d'erreur précis et le programme doit continuer avec la prochaine ligne. Il est primordial d'éviter les fuites de mémoire et les pointeurs fous. Lorsque votre programme sera testé, nous ferons varier artificiellement l'espace mémoire disponible à votre programme. Il est donc tout à fait possible que n'importe quel appel à la fonction malloc retournera NULL. Votre programme C doit seulement inclure les fichiers d'entête "`stdio.h`", "`stdlib.h`" et "`string.h`".

2 Rapport

Vous devez rédiger un rapport qui :

1. Explique brièvement le fonctionnement général du programme (maximum de 1 page au total).
2. Explique comment les problèmes de programmation suivants ont été résolus (en 2 à 4 pages au total) :
   (a) comment les nombres et variables sont représentés
   (b) comment se fait l'analyse de chaque ligne et le calcul de la réponse
   (c) comment se fait la gestion de la mémoire
   (d) comment les algorithmes d'addition, soustraction et multiplication sont implantés
   (e) comment se fait le traitement des erreurs

3 Évaluation

- Ce travail compte pour 15 points dans la note finale du cours. Indiquez vos noms clairement au tout début du programme. **Vous devez faire le travail par groupes de 2 personnes. Vous devez confirmer la composition de votre équipe (noms des coéquipiers) au démonstrateur par courriel au plus tard le 3 octobre. Si vous ne trouvez pas de partenaire d'ici quelques jours, venez me voir.**
- Le programme sera évalué sur 6 points et le rapport sur 9 points. Un programme qui plante à l'exécution, même dans une situation extrême, se verra attribuer zéro sur 6 (c'est un incitatif à bien tester votre programme). Assurez-vous de prévoir toutes les situations d'erreur (en particulier un appel à la fonction `malloc` de C qui retourne NULL et des calculs très grands).
- Vous devez remettre votre rapport (un fichier ".pdf") et le programme (un fichier ".c") au plus tard le vendredi 14 octobre à 23:59 *** sur le site Studium du cours.*** Notez que votre programme doit être contenu dans un seul fichier ".c".
- L'élégance et la lisibilité du code, l'exactitude et la performance, la lisibilité du rapport, et l'utilisation d'un français sans fautes sont des critères d'évaluation.