IFT2035 – Travail pratique #1 – 6 mai 2014

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, les pointeurs, et le traitement d'exception par code d'erreur.

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 lorsque c'est approprié).

L'application est un petit interprète pour un langage d'expressions arithmétiques. L'application lit de son entrée des expressions postfixes à raison d'une par ligne et produit en sortie, pour chaque expression, l'expression en format préfixe (syntaxe de Scheme), l'expression en format infixe (syntaxe de C), l'expression en postfixe (syntaxe de Postscript), et la valeur de l'expression. Voici un exemple d'utilisation:

```
EXPRESSION? 11 22 +         <== entré par l'usager
    Scheme: (+ 11 22)
         C: 11+22
Postscript: 11 22 add
    Valeur: 33

EXPRESSION? 3 + 4           <== entré par l'usager
ERREUR DE SYNTAXE DANS L'EXPRESSION: 3 + 4

EXPRESSION? 3 4 + 5 6 - *   <== entré par l'usager
    Scheme: (* (+ 3 4) (- 5 6))
         C: (3+4)*(5-6)
Postscript: 3 4 add 5 6 sub mul
    Valeur: -7

EXPRESSION? ^D              <== fin-de-fichier
```

La syntaxe des expressions postfixes lues par le programme est donnée par la grammaire suivante:

```
<expr> ::= <nombre> | <expr> <expr> <op>
<op> ::= "+" | "-" | "*" | "/"
<nombre> ::= <chiffre> { <chiffre> }
<chiffre> ::= "0" | "1" | "2" | "3" | "4" |
              "5" | "6" | "7" | "8" | "9"
```

À noter qu'il est permis d'avoir des espaces avant et après toute `<expr>`.

Pour chaque ligne, votre programme doit tout d'abord construire un ASA de l'expression. L'ASA doit être créé dynamiquement et il ne doit pas y avoir de limite sur la taille de l'expression. Les nombres peuvent

avoir un nombre quelconque de chiffres, et tous ces chiffres doivent être reproduits dans les expressions Scheme, C, et Postscript. Cependant, pour le calcul de la valeur de l'expression, vous pouvez utiliser des nombres point flottants (type `double`).

Utilisez la fonction `malloc` pour l'allocation mémoire. La conversion en préfixe, infixe et postfixe, ainsi que le calcul de la valeur de l'expression doit se faire en traversant l'ASA.

Votre programme doit être robuste. S'il y a un problème (mauvaise syntaxe, division par zéro, ou toute autre situation exceptionnelle) le programme doit donner un message d'erreur précis incluant l'expression qui a été lue et le programme doit continuer avec la prochaine expression. Les expressions peuvent être arbitrairement longues (votre programme ne doit donc pas imposer de limite sur la longueur de l'expression entrée par l'usager). Il est primordial d'éviter les fuites de mémoire et les pointeurs fous. Lors de l'impression en syntaxe de C, votre programme doit utiliser le minimum de parenthèses possibles. Donc on doit imprimer `(1+2)*3*4`, non pas `(((1+2)*3)*4)`.

Le format d'entrée et de sortie de l'exemple doit être respecté à la lettre car vos programmes seront testés automatiquement et le résultat sera comparé (avec l'utilitaire "`diff`") avec le résultat produit par notre programme. Faites attention tout particulièrement à des blancs superflus que votre programme pourrait imprimer. 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 ASA sont représentés
   (b) comment se fait l'analyse syntaxique, la détection des erreurs de syntaxe, et la construction de l'ASA
   (c) comment se fait la conversion en syntaxe C en minimisant les parenthèses
   (d) comment se fait le traitement des erreurs
   (e) comment se fait la récupération de l'espace mémoire

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 au plus tard à la démonstration du 15 mai. Si vous ne trouvez pas de partenaire d'ici quelques jours, venez me voir.**
- Le programme sera évalué sur 50% et le rapport sur 50%. Un programme qui plante à l'exécution, même dans une situation extrême, perdra une partie importante des points, allant jusqu'à 50% si

la nature de l'erreur le justifie. Vous devez donc accorder une grande importance à l'exécution et aux cas limites. Assurez-vous de prévoir toutes les situations d'erreur (en particulier un appel à la fonction `malloc` de C qui retourne `NULL`).

- Vous devez remettre un fichier ".tar" qui contient uniquement deux fichiers : votre rapport (qui doit se nommer "rapport.pdf") et le programme (qui doit se nommer "tp1.c"). La remise doit se faire au plus tard à 21h lundi le 2 juin sur le site Studium du cours. En supposant que vos deux fichiers sont dans le répertoire "tp1", vous pouvez créer le fichier ".tar" avec la commande "`tar cf tp1.tar tp1`".
- 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.