Programmation Impérative - TP 4 & 5
Une autre calculatrice

Rohan Fossé - Léo Mendiboure - Guillaume Mercier
{rohan.fosse,leo.mendiboure}@labri.fr, mercier@enseirb-matmeca.fr

2018-2019

1 Présentation

L'objectif de ce TP est d'implémenter une calculatrice qui utilise la notation préfixe, c'est à dire que l'opérateur est placé *avant* ses opérandes. La notation préfixe permet de se passer de parenthèses. Par exemple, avec un tel système, l'opération : 2 + 3 (notation infixe) sera notée : + 2 3 (notation préfixe). Quand on trouve un opérateur (i.e. +,−,∗ et /), il faut procéder au calcul de l'opération en regardant la suite des arguments et procéder à leur évaluation. Il y a deux cas :

- soit on trouve un nombre et l'évaluation est directe : c'est la valeur de ce nombre
- soit on trouve un nouvel opérateur et on recommence le processus d'évaluation en regardant les arguments suivants. Il y a donc une notion de *récursion* implicite au calcul.

Dans un premier temps, nous allons formuler les hypothèses suivantes :

- les arguments du programme lui seront passés via la *ligne de commande*. Par exemple : `calcul_pref * 2 + 3 4` (qui renverra comme résultat `14`).
- la suite d'arguments passée au programme est *correcte*, c'est à dire qu'il n'est pas nécessaire de gérer le cas où l'utilisateur fait une erreur de saisie quand il rentre une suite d'opérations
- les arguments passés au programme sont de deux types uniquement : des nombres ou bien des opérateurs (c.f liste ci-dessus). Cas particulier : l'opérateur de multiplication `*` est interprété d'une façon non convenable par le shell, il faudra donc mettre des doubles guillemets (donc `"*"` à la place de `*`).
- les nombres sont des entiers positifs ou nuls (pour le moment)
- les opérateurs sont tous binaires (pour le moment)

En ce qui concerne l'implémentation de cette calculatrice, nous allons utiliser une structure d'*arbre binaire*, qui est une généralisation de la structure de liste chaînée déjà vue et manipulée. Une telle structure va se baser sur la définition d'un nouveau type de données, par exemple :

```
typedef struct tree{
  int value;          /* valeur de l'opérande */
  char operator;       /* opérateur */
  struct tree *left;  /* sous-arbre gauche */
  struct tree *right; /* sous-arbre droit */
} node_t;
```

Chaque argument passé au programme sera donc stocké et manipulé par ce dernier sous la forme d'une variable de type `node_t`. Un élément de l'arbre possède donc une *étiquette* et deux pointeurs sur des *sous-structures*. Cette étiquette est soit une valeur (un nombre, le champ `value` de la structure de type `node_t`), soit un opérateur arithmétique (le champ `operator`). Ces deux éléments sont mutuellement exclusifs car un argument est soit un nombre (une feuille de l'arbre), soit un opérateur (un nœud interne de l'arbre). Enfin, une variable de type `node_t` sera utilisée pour stocker une référence sur la *racine* de l'arbre. On accédera à l'arbre uniquement via cette variable. En pratique, il va s'agir d'un pointeur sur une structure de type `node_t`. Par exemple :

```
node_t *root = NULL;
```

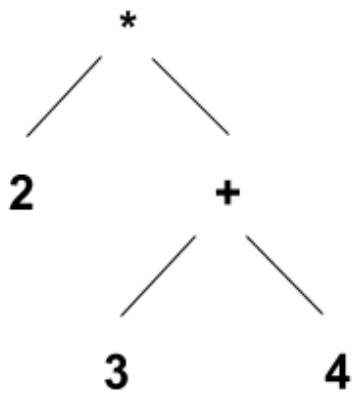
2 Implémentation

Question
Écrivez une fonction qui initialise la structure d'arbre correspondant à la suite d'opérations passée en argument au programme. Il faut donc mettre en place le chaînage de tous les élements de type `node_t` qui stockent les arguments passés au programme. Rappel : on ne passe pas la suite d'opérations à évaluer sous la forme d'une unique chaîne de caractères, mais chaque nombre ou opérateur est une chaîne distincte (par ex, on lance `calcul_pref "*" 2 + 3 4` et non pas `calcul_pref "* 2 + 3 4"`). Un prototype possible pour cette fonction pourra être :

```
node_t *build_tree(char *argv[],int *position);
```

Le paramètre `position` pourra servir à mémoriser la position courante dans la ligne de commande que le programme est en train de traiter (i.e le numéro de l'argument (opérande ou opérateur) que le programme est en train de traiter).
Notes :

1. `$ ./calcul_pref "*" 2 + 3 4` devra conduire à la construction de l'arbre présenté ci-dessous, arbre binaire dans lequel chaque opérateur possède deux sous-arbres :



2. La fonction de construction de l'arbre `build_tree` pourra s'appuyer sur la récursivité intrinsèque de la structure de données.
3. L'utilisation de fonctions telles que `isdigit` ou `atoi` pourra se révéler pertinent ...

Question
Écrivez une fonction qui transforme la suite d'opération initiale en mode préfixe et l'affiche en mode infixe. L'affichage en mode infixe s'obtient par un parcours *en largeur d'abord* de la structure de donnée arborescente. Vous pourrez vous appuyer sur la récursivité intrinsèque de la structure de données choisie pour implémenter la calculatrice :

- quand on trouve un nombre (i.e. une feuille de l'arbre), on l'affiche directement à l'écran
- quand on trouve un opérateur (i.e un nœud interne de l'arbre) on va afficher le sous-arbre de gauche puis l'opérateur et enfin le sous-arbre de droite (utilisez la récursivité). Les opérateurs étant tous binaires, ces sous-arbres sont forcément non vides.

Un prototype possible pour cette fonction pourra être :

```
void display_tree(node_t *root);
```

Note : vous êtes invités à rajouter des parenthèses à l'affichage afin d'éviter les ambiguïtés du mode infixe et respecter l'associativité des opérations. Par exemple, `$ ./calcul_pref "*" 2 + 3 4` retournera `(2 * ( 3 + 4))`.

Question
Écrivez une fonction qui effectue le calcul proprement dit de la suite d'opérations passée en arguments. Ce calcul se fera en effectuant un parcours *en profondeur d'abord* de la structure arborescente. Là encore, vous vous aiderez de la récursivité intrinsèque de la structure de données :

- quand on trouve un nombre (une feuille), l'évaluation est directe puisqu'il s'agit de la valeur de ce nombre. La fonction retourne donc ce nombre.
- quand on trouve un opérateur (une nœud), on va évaluer le sous-arbre de gauche, puis celui de droite. Ces deux sous-évaluations vont produire deux valeurs qui seront les opérandes de l'opération que l'on est en train de traiter.

Un prototype possible pour cette fonction pourra être :

```
int compute_tree(node_t *root);
```

Par exemple, `$ ./calcul_pref "*" 2 + 3 4` retournera 14. En effet, l'évaluation du sous-arbre de gauche renverra 2 et celle du sous-arbre de droite renverra de façon récursive $4 + 4 = 7$, la multiplication de 7 par 2 donnant 14.

3 Améliorations

Question
Modifiez votre programme pour obtenir une calculatrice qui permet de travailler avec des entiers négatifs.

Question
Modifiez votre programme pour obtenir une calculatrice qui permet de travailler avec des nombres flottants.

Question
Modifiez votre programme pour n'utiliser que des pointeurs et se passer totalement de tableaux.

Question
Définissez et implémentez une fonction permettant de transformer l'expression préfixe en une impression postfixe et de l'afficher à l'écran. Par exemple,
`$ ./calcul_pref "*" 2 + 3 4` retournera `3 4 + 2 *` et
`$ ./calcul_pref * + 1 2 - 3 4` retournera `1 2 + 3 4 - *`.