TRAVAUX PRATIQUES X

Calculatrice en notation polonaise inverse

La notation mathématique usuelle pour les calculs est ce que l'on nomme une ntoation infixe. Par exemple, dans $x + y$ les arguments de la fonction + sont écrits à gauche et à droite du nom de la fonction. Quand ovous travaillez sur des fonctions (en informatique ou en mathématiques), vous avez plutôt l'habitude de la notation préfixe : $plus(x, y)$; ou encore $+\ x\ y$.

La notation polonaise inverse (**R**everse **P**olish **N**otation, ou **RPN**) est basée sur l'écriture postfixe : $x\ y\ +$. Elle a l'avantage d'éviter les ambiguïtés de parenthésage [1], et d'offrir un algorithme très simple de calcul de l'expression. Elle est aussi un élément très important de l'histoire de l'informatique, ayant été populaire dans les calculatrices des années 60-70. Elle permettait aussi un débogage plus agréable [2].

A Notation polonaise inverse

A.1 Introduction

Traduisons un exemple de RPN : « 3 1 + 2 − 3 × ». Pour calculer une telle expression, c'est en fait assez simple : on va lire la notation postfixe de gauche à droite, en maintenant une *pile* des valeurs déjà calculées. Lorsque l'on croise une valeur dans la notation, on l'empile. Lorsque l'on croise une fonction à k arguments, on dépile les k premières valeurs de la pile, on les passe à la fonction, puis on empile le résultat. À la fin, le résultat est au sommet de la pile.

Ainsi, à tout moment la pile contient les résultats des calculs intermédiaires précédents qui n'ont pas encore été utilisés (ceux qui seront utilisés le plus tôt sont au sommet de la pile). Notez qu'il n'y a jamais besoin de revenir en arrière dans la lecture de l'expression !

FIGURE X.1 – Évolution de la pile lors du calcul de 3 1 + 2 − 3 ×

Voici un pseudo-code (récursif) de l'évaluation d'une expression :

Algorithme 4 : calcule

Entrées : *instr* une suite d'instruction (en RPN), *pile* une pile (les résultats des calculs intermédiaires précédents)

```
si instr est vide alors
    renvoyer le sommet de pile
sinon si la première instruction de instr est un entier n alors
    Empiler n sur pile
    CALCULE(instructions suivantes, pile)
sinon si la première instruction de instr est une fonction f alors
    k ←nombre d'arguments de f
    Dépiler x_1, x_2, ..., x_k de pile
    Empiler f(x_1, ..., x_k) sur pile
    CALCULE(instructions suivantes, pile)
sinon
    Impossible (on peut par exemple lever une exception)
```

1. La notation préfixe le permet aussi, mais les langages informatiques vont rarement jusque là : ils implémentent un mélange de préfixe et infixe, qui correspond à ce que l'on fait en maths.
2. Pour en savoir plus : `https://fr.wikipedia.org/wiki/Notation_polonaise_inverse`

A.2 Plus avancé

Le calcul infixe $(1 + 2) \times (3 + 4)$ s'écrit : « 1 2 + 3 4 + × ». Pour le calculer, on applique le même algorithme que précédemment :

FIGURE X.2 – Évolution de la pile lors du calcul de 1 2 + 3 4 + ×

1. Calculez à la main le résultat de :
   a. 0 2 − 5 6 + ÷
   b. 1 4 3 × +
2. Traduisez en notation polonaise inversée, puis évaluez :
   a. $50(8 - 3 + 20) - 2(2.7 + 8(4 - 5))$
   b. $\frac{2.3^2 - 7.3 + 2}{4.3 - 20}$

B En OCaml

Variante étoilée du TP : n'utilisez pas le code à trous fourni.

En guise de pile, nous utiliserons une liste. Empiler revient à ajouter un élément en tête de liste, dépiler à lire l'élément de tête puis ne travailler qu'avec la queue ensuite.

Les instructions seront une liste de `string`. Chaque élément de cette liste soit correspond à une fonction (`"+"` , `"-"` , etc) soit à un entier (`"12"`).

3. Traduisez le pseudo-code en une fonction `calcule : string list -> int list -> int` .

   Vous pouvez utiliser toutes les fonctions du modules `List` qui vous intéressent, et notament `List.hd` (renvoie la tête d'une liste) et `List.tl` (renvoie la queue d'une liste).

   Vous pouvez utiliser `int_of_string` pour convertir une string en entier.

4. En déduire une fonction `calculatrice : string list -> int` qui prend en argument une RPN et renvoie le résultat du calcul.

   Pour tester votre fonction, utilisez `utop` . Pour créer des entrées, vous pouvez :

   - soit écrire la RPN à la main.
   - soit utiliser `String.split_on_char ' '` : elle prend une `string`, la coupe en plusieurs string en séparant sur les espaces, et renvoient la liste des strings obtenus. Ainsi, `String.split_on_char ' ' "50 3 +"` vaut `["50"; "3"; "+"]`

5. Faites en sorte que la fonction lève l'exception `Invalid_argument "calculatrice"` si jamais une des instructiosn n'est ni une fonction ni un entier.
6. Ajoutez le modulo aux opérations. Je vous conseille d'utiliser `%` pour le représenter.
7. (Terminale NSI) : en termes d'arbres, à quoi correspondre les notations préfixes, infixe et postfixe ? Faire des schémas pour expliquer.
8. (Difficile, mi-HP) : regardez la fonction `List.fold_left` , et utilisez-la pour remplacer `calcule` .