Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 49 | analyse d'expressions

Présentation

Le but de ce TP est de manipuler les piles de données vues en cours. Pour cela, on s'intéressera à l'analyse d'expressions représentant des calculs arithmétiques simples.

1 Notation polonaise inversée

La notation la plus répandue pour écrire une expression arithmétique est la notation *infixée* : un opérateur binaire est placé entre ses deux opérandes : $1 + 2$. Cette notation nécessite parfois l'utilisation de parenthèses pour exprimer certains calculs, comme par exemple : $(1 + 2) \times (3 + 4)$.

Il existe également des notations dites *préfixées* et *postfixées*. Avec une notation préfixée, l'opérateur précède ses opérandes $(+1\ 2)$ alors qu'il leur succède si on utilise une notation postfixée $(1\ 2\ +)$.

La *notation polonaise* est une notation préfixée proposée par le mathématicien polonais Jan Łukasiewicz en 1920. Dans cette notation, chaque opérateur possède une arité bien définie : il n'existe pas d'opérateurs homonymes. Par exemple, l'opérateur soustraction (binaire) n'est pas représenté par le même signe que l'opérateur négation (unaire). La notation polonaise offre la particularité de ne pas nécessiter de parenthèses : l'expression infixée $(1 + 2) \times (3 + 4)$ devient : $\times + 1\ 2 + 3\ 4$.

La notation polonaise inversée (NPI) est une version postfixée de la notation polonaise. L'avantage sur la notation polonaise est que l'algorithme qui permet d'évaluer des expressions NPI est plus simple. C'est pour cette raison que les premières calculatrices électroniques (dans les années 60) utilisaient cette notation. Avec la NPI, l'expression précédente prend la forme suivante : $1\ 2 + 3\ 4 + \times$.

2 Méthode d'évaluation

Pour évaluer une expression écrite en NPI, on utilise une pile de données et on parcourt l'expression en partant de la gauche et en traitant chaque *signe*. Ici, le mot *signe* désigne un élément quelconque de l'expression, c'est-à-dire un opérande ou un opérateur.

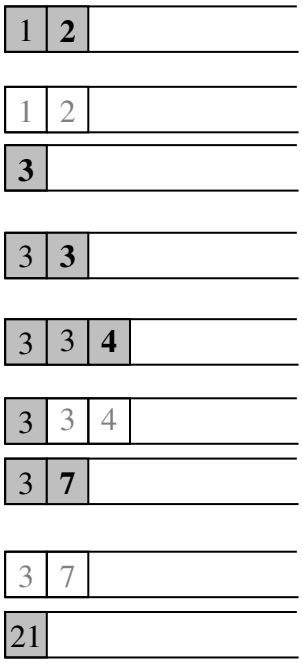
- Tant qu'il reste des signes dans l'expression :
  - Si le signe est un *opérande* :
    - Il est empilé.
  - Si le signe est un *opérateur* :
    - On dépile un nombre d'opérandes correspondant à son arité ;
    - On applique l'opérateur à ces opérandes ;
    - Le résultat est empilé.
- S'il ne reste plus de signe dans l'expression, alors le *sommet* de la pile correspond au *résultat* de l'évaluation.

exemple : évaluation de l'expression $1\ 2 + 3\ 4 + \times$

- **1** $2 + 3\ 4 + \times$:

1	

    - On empile 1.
- 1 **2** + 3 4 +× :
    - On empile 2.
- 1 2 **+** 3 4 +× :
    - On dépile deux opérandes : 2 et 1.
    - On calcule 1 + 2.
    - On empile le résultat 3.
- 1 2 + **3** 4 +× :
    - On empile 3.
- 1 2 + 3 **4** +× :
    - On empile 4.
- 1 2 + 3 4 **+**× :
    - On dépile deux opérandes : 4 et 3.
    - On calcule 3 + 4.
    - On empile le résultat 7.
- 1 2 + 3 4 +**×** :
    - On dépile deux opérandes : 7 et 3.
    - On calcule 3 × 7.
    - On empile le résultat 21.
- 1 2 + 3 4 +× :
    - Il n'y a plus de signe dans l'expression.
    - Le résultat est le sommet : 21.



3 Implémentation de la méthode

Nous manipulerons exclusivement des opérandes entiers. L'expression à évaluer sera représentée par une chaine de caractères pouvant contenir les caractères suivants :

- Les opérateurs :
    - `'+'` (addition).
    - `'-'` (soustraction).
    - `'*'` (multiplication).
    - `'/'` (division entière).
    - `'%'` (reste de la division entière).
- Les chiffres de `'0'` à `'9'`.
- Le caractère espace `' '`.

Le caractère espace est utilisé seulement pour séparer deux opérandes qui ne sont pas déjà séparés par un opérateur. Le caractère espace ne doit apparaître que dans ce cas bien précis. En particulier, les opérandes et les opérateurs ne sont pas séparés par un espace.

exemples : l'expression 1 2 + 3 4 +× :

- est représentée par la chaîne `"1 2+3 4+*"`.
- n'est *pas* représentée par la chaîne `"1 2 + 3 4 + *"`.

L'archive fournie avec ce sujet contient une bibliothèque `pile_liste` correspondant à l'implémentation du type abstrait *pile* au moyen d'une liste. La bibliothèque contient également les fonctions de manipulation des piles vue en cours. L'archive contient aussi la bibliothèque `liste_s` (liste simplement chaînée), qui est utilisée par `pile_liste`.

Exercice 1

Écrivez une fonction `int est_operateur(char c)` qui reçoit un caractère `c`. La fonction doit renvoyer 1 si le caractère passé en paramètre est un opérateur, ou 0 sinon.

exemples :

- Pour `c='1'`, la fonction doit renvoyer 0.
- Pour `c=' '`, la fonction doit renvoyer 0.
- Pour `c='*'`, la fonction doit renvoyer 1.

Exercice 2

Écrivez une fonction similaire `int est_chiffre(char c)` qui reçoit un caractère et détermine si le caractère est un chiffre.

Exercice 3

Écrivez une fonction `int calcule_entier(char *exp, int *pos)` qui reçoit l'expression `exp` et le numéro `pos` d'un de ses caractères. La fonction doit calculer la valeur du nombre entier dont les chiffres sont compris entre le caractère numéro `pos` (inclus) et le prochain caractère qui ne soit pas un chiffre. Elle doit modifier `pos` de manière à ce qu'il indique ce caractère de séparation. Enfin, elle doit renvoyer la valeur de l'entier.

Remarque : on suppose que le caractère initialement indiqué est un chiffre.

exemples : la fonction reçoit l'expression suivante :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
9	9	4		3	2	4	+	5	4	2	+	3		4	+	*	\0

- Pour `*pos=0` : le résultat est `994` et `*pos` a la valeur `3` à la fin du traitement.
- Pour `*pos=8` : le résultat est `542` et `*pos` prend la valeur `11`.
- Pour `*pos=14` : le résultat est `4` et `*pos` prend la valeur `15`.

Exercice 4

Écrivez une fonction `int applique_operateur(char opt, int opd1, int opd2)` qui renvoie le résultat du calcul consistant à appliquer l'opérateur représenté par le caractère `opt` aux deux opérandes `opd1` et `opd2` passés en paramètres.

Remarque : on suppose que le caractère `opt` représente bien un opérateur.

exemple : `applique_operateur('+',2,3)` doit renvoyer la valeur *5*.

Exercice 5

En utilisant les fonctions précédentes et les fonctions de manipulation de piles, écrivez une fonction `int evalue_NPI(char *expression, int *resultat)` qui évalue une expression écrite en NPI. Vous appliquerez l'algorithme décrit plus haut.

Le résultat de l'évaluation sera renvoyé par adresse via le paramètre `resultat`. En cas d'erreur, la fonction doit renvoyer -1 par valeur. En cas de succès, elle renvoie 0. Une erreur se produit quand l'expression passée en paramètre ne respecte pas les règles décrites précédemment.

4 Gestion des erreurs

Exercice 6

Écrivez une fonction `void interface_NPI()` qui :

- Demande à l'utilisateur de saisir une expression
- Évalue l'expression
  - En cas de succès : la fonction affiche le résultat.
  - En cas d'erreur : la fonction affiche un message d'erreur.

exemples :

- Une expression correcte :

```
Entrez l'expression NPI a evaluer : 1 2+3 4+*
Le resultat est : 21.
```

- Une expression incorrecte :

```
Entrez l'expression NPI a evaluer : 1 2 +
Erreur lors de l'evaluation.
```

Exercice 7

Le message d'erreur de la fonction précédente est imprécis, ce qui rend difficile la correction de l'éventuelle erreur détectée dans une expression. Faites une copie de `evalue_NPI` que vous appellerez `evalue_NPI_2`, et modifiez-la pour qu'elle renvoie différents codes d'erreur :

- La chaîne représentant l'expression est vide : -6
- La chaîne contient un caractère interdit : -5
- La chaîne contient un espace mal placé : -4
- Il manque un opérateur dans la chaine : -3
- Il manque un opérande dans la chaine : -2
- Une autre erreur survient (par exemple : erreur d'accès mémoire dans une fonction de la pile) : -1
- Il n'y a pas d'erreur : 0

Ces différentes valeurs doivent être définies sous formes de constantes.

Exercice 8

Faites une copie de `interface_NPI` que vous appellerez `interface_NPI_2` et modifiez-la de manière à lui faire afficher un message différent pour chaque code d'erreur possible.

exemple :

```
Entrez l'expression NPI a evaluer : 1 2+3 4+
Erreur : un operateur est manquant.
```

Exercice 9

Faites des copies des fonctions `evalue_NPI_2` et `interface_NPI_2` que vous appellerez respectivement `evalue_NPI_3` et `interface_NPI_3`. Modifiez-les de manière à faire apparaître la position du caractère de la chaîne qui a provoqué l'erreur détectée.

exemple :

```
Entrez l'expression NPI a evaluer : 1 2 +
Erreur : un espace est mal place (pos.3).
```

Exercice 10

Écrivez une fonction `void souligne_erreur(char *expression, int position)` qui affiche l'expression en la soulignant jusqu'à la position passée en paramètre.

exemple : pour l'expression et la position de l'erreur de l'exemple précédent

```
   1 2 +
   ^^^^
```

Exercice 11

Faites une copie de la fonction `interface_NPI_3` que vous appellerez `interface_NPI_4`, et modifiez-la de manière à lui faire afficher l'expression soulignée en cas d'erreur.

exemple :

```
Entrez l'expression NPI a evaluer : 1 2+3 4+
Erreur : un operateur est manquant (pos.8).
   1 2+3 4+
   ^^^^^^^^
```