POLYTECH SORBONNE

Sorbonne Université
2021-2022

Cours de Programmation en C – ROB3

Travaux Pratiques

Objectif(s)

- Liste chaînée
- Allocation dynamique de mémoire

1 Introduction

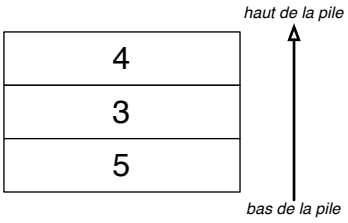
La notation polonaise inverse (en anglais *RPN* pour *Reverse Polish Notation*), également connue sous le nom de notation post-fixée, permet de noter les formules arithmétiques sans utiliser de parenthèses. Elle est utilisée notamment dans les calculatrices scientifiques HP et les langages informatiques *PostScript*[1] et *Forth*[2], etc. Elle consiste, pour les opérations, à écrire les opérandes **avant** l'opérateur.

La réalisation de calculatrices RPN est basée sur l'utilisation d'une pile; c'est-à-dire, que les opérandes sont ajoutées en haut de la pile, et les résultats des calculs sont retournés en haut de la pile. Bien que ce concept puisse sembler inutilement compliqué au début, l'analyse d'une expression sous forme RPN a l'avantage de la concision. Sur un ordinateur, elle se prête d'ailleurs bien aux transformations en raison de sa grammaire simple.

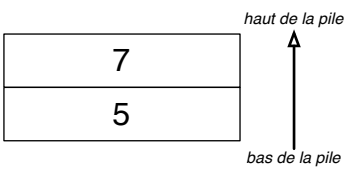
Par exemple, pour calculer $(3+4)*5$, on écrit 3 4 + puis 5 $*$. Il peut aussi s'écrire 5 3 4 + $*$.

Au fur et à mesure de la saisie des valeurs, ceux-ci sont placés dans une pile. Une pile (en anglais *stack*) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (ou LIFO pour *Last In, First Out*), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés. Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes en haut de la pile, et on les récupère toujours sur le haut de la pile (donc dans l'ordre inverse où on les a posé, c'est-à-dire en commençant par la dernière ajoutée).

Lorsque l'on saisit des valeurs, celles-ci sont empilées. Par exemple, quand on saisit "`5 3 4`", le 5 est ajouté à la pile vide, puis le 3 puis le 4. On obtient donc la pile suivante:



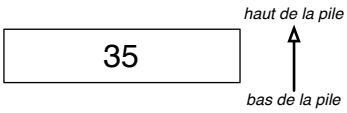
Puis, lorsque le "+" est traité, les deux 1ères valeurs de la pile (4 et 3) sont retirées (on dit *dépilées*), additionnées, puis le résultat est rajouté (ou *empilé*) dans la pile. On obtient donc la pile suivant:



[1] Langage de description de page : `http://fr.wikipedia.org/wiki/PostScript`

[2] Langage bas-niveau de programmation, notamment utilisé depuis les années 80 dans les systèmes embarqués, notamment à la NASA : `http://fr.wikipedia.org/wiki/Forth_(langage)` et `http://forth.gsfc.nasa.gov/`

Enfin, lorsque le "$\star$" est traité, on obtient une pile avec une seule valeur, qui est le résultat du calcul:



Si l'on rajoute quelques opérations pour manipuler des éléments de la pile (duplication du 1er élément, échange des deux 1ers éléments, etc.), on obtient tous les éléments nécessaires pour réaliser efficacement tous les calculs possibles.

Le but du TP est donc de réaliser un mini-interpréteur à pile pour calculer des opérations.

2 Pile

Pour commencer, on veut pouvoir manipuler une pile de réels. Une telle pile va être réalisée grâce à une liste (simplement) chaînée pour laquelle on ne manipulera que le 1er élément. Rajouter un élément au sommet de la pile reviendra donc à insérer un élément en début de liste. Enlever un élément au sommet de la pile revient donc à supprimer le 1er élément de la liste.

Et puisqu'il ne nous sera nécessaire que d'insérer ou d'enlever des éléments en début de liste (en haut de la pile), un pointeur vers le 1er élément de la liste/pile sera suffisant pour représenter la liste/pile (pas besoin de la structure que l'on a vu en TD). Toutes les fonctions manipulant la pile (empiler, dépiler, etc.) renverront un pointeur vers le (nouveau) 1er élément de la pile. De plus, une pile vide sera donc représentée par un pointeur `NULL`;

Question 1
Dans le fichier `pile.h` définir une structure `t_element` pour représenter un élément de la pile de réels.

Question 2
Dans un fichier `pile.c`, écrire une fonction `t_element* empiler(t_element* sommet, double reel)` qui empile un élément au sommet de la pile.

Question 3
Écrire une fonction `t_element* depiler(t_element* sommet)` qui enlève l'élément au sommet de la pile. Gérer le cas de la pile vide.

Question 4
Écrire une fonction `t_element* vider(t_element* sommet)` qui vide la pile (en appelant successivement la fonction `depile`).

Question 5
Écrire une fonction `void afficher(t_element* sommet)` qui affiche le contenu de la pile. Par exemple, on cherchera à afficher quelque chose comme :

```
Pile:
  12.02
  3
  6.58
```

Question 6
Dans un fichier `mainP1.c`, écrire un petit programme ajoutant quelques valeurs dans la pile, affichant la pile, et les supprimant avant de quitter.

3 Saisie au clavier

On veut maintenant saisir au clavier des chaînes de caractères. S'ils correspondent à un réel, il faut ajouter ce réel à la pile, sinon, il faut évaluer la chaîne de caractères pour décider d'une action à mener sur la pile.

Question 7
Dans un fichier `mainP2.c`, écrire un programme principal qui saisit des chaînes de caractères, l'ajoute à la pile si c'est un réel (l'ignore sinon) et affiche la pile. La saisie des chaînes devra s'arrêter quand on tape la chaîne "`EXIT`", et la pile devra être vidée avant de quitter le programme.

Pour la conversion d'une chaîne de caractères en flottant double, on utilisera la fonction `strtod` qui définie dans `string.h`. Elle permet de convertir une chaîne de caractères en `double` de manière plus robuste que la fonction `atof`. Un exemple d'utilisation (à adapter) est le suivant:

```
char* a_convertir = "une chaine...";
char* p;
double val = strtod(a_convertir, &p);
if (*p != '\0')
    printf("conversion non possible!\n");
else
    printf("la valeur est %f",val);
```

Pour la comparaison de chaînes de caractères, c'est bien sûr la fonction `strcmp` qu'il faut utiliser.
Vous trouverez plus d'informations avec les commandes `man strtod` et `man strcmp`.

Tester le programme avec "`12 342.23 TOTO 56 EXIT`".
Avant de quitter, l'affichage de la pile devrait être le suivant :

```
Pile:
  56.000000
  342.230000
  12.000000
```

4 Opérations arithmétiques

On veut maintenant pouvoir réaliser des opérations mathématiques sur les éléments de la pile. Lorsque l'on saisira un opérateur mathématique (+, −, `SIN`, ...), les opérandes nécessaires seront dépilés de la pile, utilisés pour l'opération, et le résultat sera rajouté sur la pile. Par exemple, l'opérateur + additionnera les 2ers nombres de la pile (ces deux nombres seront enlevés de la pile, et le résultat sera rajouté dans la pile). Si la pile vaut $\begin{array}{|c|}\hline 3\\ 8\\ 12\\ \hline\end{array}$, alors après l'addition, elle vaudra $\begin{array}{|c|}\hline 11\\ 12\\ \hline\end{array}$.

Si la pile ne contient pas assez d'opérandes pour l'opération, alors les opérandes inutilisés sont remis dans la pile et l'opération n'est pas réalisée.

Question 8
Écrire une fonction `t_element* addition(t_element* sommet)` qui dépile les 2ers éléments de la pile et empile le résultat de l'addition. S'il y a un ou zéro éléments, alors l'addition n'a pas lieu.

Question 9
Écrire une fonction `t_element* action(t_element* sommet, char* saisie)` que l'on appellera depuis le `main` lorsque la conversion de la chaîne en réel n'est pas possible. Cette fonction devra appeler la fonction `addition` si les chaînes de caractères "`ADD`" ou "+" lui sont passées.

Question 10
À partir de la fonction d'addition, écrire les fonctions de soustraction ("−" et "`SUB`"), multiplication ("*" et "`MUL`") et division ("/" et "`DIV`"; ici, le 2ème opérande ne doit pas être nul, sinon l'opération n'est pas réalisée).

Question 11
Écrire une fonction `t_element* sinus(t_element* sommet)` qui calculera le sinus du 1er élément de la pile, et qui sera appelée avec la chaîne de caractère "`SIN`" (la fonction `sin` de la bibliothèque standard du C est définie dans `math.h`).

Calculer le résultat de l'expression "`12 45 * 36 + 25 47 / +`". Comment écrire l'opération $(12+35)*\left(\frac{86+0.9}{35}+\frac{56}{87+25}\right)$?

5 Opérations sur la pile

Il est aussi important de pouvoir avoir des opérateurs sur la pile, notamment pour échanger les 2ères valeurs, pour en dupliquer une, etc...

Question 12
Écrire une fonction `t_element* dupliquer(t_element* sommet)` (dans `pile.c`) qui duplique la 1ère valeur de la pile. Modifier la fonction `action` pour que la 1ère valeur de la pile soit dupliquée si on saisie la chaîne "`DUP`".

Question 13
Écrire une fonction `t_element* echanger(t_element* sommet)` qui échange les 2 premières valeurs de la pile. L'associer avec la chaîne "`SWAP`".

Question 14
La saisie de la chaîne "`POP`" doit retirer la 1ère valeur de la pile.

Question 15
La saisie de la chaîne "`CLEAR`" doit vider la pile.

Les fonctions "`DUP`" et "`SWAP`" sont utiles pour réutiliser plusieurs fois un résultat dans un calcul. Les utiliser pour calculer $\left(\frac{28+35}{(5+12)*(6+7)}+12\right)*sin\left(\frac{28+35}{(5+12)*(6+7)}\right)$ (en calculant la valeur intermédiaire $\frac{28+35}{(5+12)*(6+7)}$ et la dupliquant pour l'utiliser ensuite).

6 Bonus

Question 16
Dans toutes fonctions manipulant la pile, il faut écrire `sommet = maFonction( sommet, ...);`. Proposez une modification (avec un nouveau type ou en utilisant le type `t_element`) permettant d'écrire ces appels sous la forme `maFonction( pile, ...);`

Question 17
Rajouter une fonction `haut`, associée à la chaîne "`UP`" qui décale tous les éléments de la pile vers le haut, et met le 1er élément au bas de la pile.

Question 18
Rajouter une fonction `bas` associée à la chaîne "`DOWN`" qui décale tous les éléments vers le bas, et met le tout dernier élément de la pile tout en haut.