Programmation C avancée
TP 3

L'objectif de ce TP est de faire des rappels sur les structures chaînées et d'affiner un peu sur les mécanismes évolués d'entrées sorties.

On veut réaliser une légère calculatrice utilisant les expressions algébriques postfixées (communément désignées par expressions en Polonais inversé). Cela consiste à manipuler une pile (dernier entré, premier sorti) dont les objets seront pour nous des entiers. Lorsque le programme démarre, la pile est vide et l'interpréteur attend des interactions de l'utilisateur.

L'utilisateur doit alors donner soit des nombres, soit des opérateurs, soit solliciter une fonctionnalité. Toute entrée doit être séparée par un délimiteur (l'utilisateur utilise un espace ou le retour chariot). Lorsqu'un nombre est lu, ce dernier est empilé sur la pile courante du programme. Si un opérateur binaire est lu (comme +, -, *, /, %), le programme utilise les deux dernières valeurs entrées, les détruit, calcule le résultat de l'opération et empile ce résultat.

Ainsi, en partant d'une pile vide, si l'utilisateur donne 2, la pile contient 2. Si l'utilisateur donne ensuite 3, la pile aura alors 3 au dessus de 2. Enfin si l'utilisateur donne un +, le 3 et le 2 sont dépilés, puis 2 + 3 est calculé et le résultat 5 est empilé. La pile ne contient donc plus que le chiffre 5.

Voici un exemple calculant la factorielle de 5 qu'avec des multiplications, on commence à donner une pile d'éléments puis on donne quatre fois l'opération de multiplication. Le `p` final demande l'affichage de la tête de la pile.

```
nborie@perceval:~/ > ./a.out
1 2 3 4 5
*
*
*
*
p
120
q
nborie@perceval:~/ >
```

Voici un autre exemple, le calcul de $(2 + 3) * (6 - 4) + 7$:

```
nborie@perceval:~/ > ./a.out
2 3 + 6 4 - * 7 +
p
17
q
nborie@perceval:~/ >
```

Amusez-vous un peu avec l'utilitaire `dc` de `Unix` pour fixer vos idées. On veut exactement le même comportement mais sans la précision arbitraire que `dc` supporte. **Nous ferons ainsi tout avec des entiers** (des `int` plus précisément).

Liste des opérations à implanter :

- addition : `+`
- soustraction : `-`
- multiplication : `*`
- division (des entiers) : `/`
- modulo : `%`
- factorielle : `!` (opérateur unaire!)
- exponentiation : `^`

Outre les nombres et les opérations, il faut aussi implanter quelques fonctionnalités que l'on peut appeler dans l'interpréteur :

- quitter le programme : `q` (quit)
- afficher la valeur en tête de pile : `p` (print)
- vider la pile : `c` (clear)
- affiche toute la pile : `a` (all)
- change l'ordre des deux valeurs en tête de pile : `r` (reverse)

Lorsqu'une commande n'est pas applicable, vous devrez écrire **le message d'erreur** relevant sur la sortie standard d'erreur (exemple : pile vide, opérandes manquantes, ...).

La grande difficulté de ce TP (par rapport à ce que vous avez l'habitude de faire) est qu'il faut lire les interactions de l'utilisateur alors que l'on ne sait pas, a priori, ce qu'il donne au programme (un nombre, un opérateur ou une fonctionnalité).

Pour lire ligne par ligne les interactions utilisateurs, vous devrez utiliser la bibliothèque `GNU readline`. Cette dernière est installée sur les machines de la faculté et pour l'avoir sur vos machines personnelles utilisant une distribution Debian-like : `sudo apt-get install libreadline-dev`. Une fois installée, voir le manuel : `man 3 readline`. Vous devrez alors compiler avec le flags `-lreadline` (à mettre en fin de ligne de compilation!).

(regarder aussi les fonctions `add_history`, `clear_history` dont les prototypes sont `void add_history(const char*)` et `void clear_history(void)`).