TP : calculatrice en notation RPN

Le but de ce TP est d'écrire un programme complet en langage C réalisant une calculatrice en notation polonaise inversée (*Reversed Polish Notation*). Il s'agit de prendre en entrée des expressions arithmétiques en utilisant non pas la notation infixe habituelle des opérateurs, mais la notation postfixe. Ainsi on n'écrira pas

```
(13 + 12) x (2 + 3)
```

mais plutôt

```
13 12 + 2 3 + x
```

un avantage est qu'il n'est plus nécessaire d'utiliser les parenthèses.

Il existe des modèles réels de calculatrices fonctionnant selon ce principe comme la *HP 35* sortie en 1972 et surnommée "règle à calcul électronique".

Notre calculatrice prendra la forme d'un programme exécutable `calc` prenant en paramètres les éléments de l'expression arithmétique à calculer. On l'utilisera ainsi dans le terminal :

```
./calc 13 12 + 2 3 + x 2 x
Résultat : 250
```

Pour réaliser une telle calculatrice, on utilise habituellement une pile LIFO ainsi :

- lorsqu'on lit un entier on le stocke dans la pile
- lorsqu'on lit un `+` on récupère les deux valeurs en sommet de pile, on les additionne et on stocke le résultat dans la pile
- lorsqu'on lit un `x` on récupère les deux valeurs en sommet de pile, on les mulitplie et on stocke le résultat dans la pile
- ...

À la fin de la lecture de l'expression, si elle est correcte, la pile ne contiendra plus qu'une seule valeur qui est le résultat du calcul. Pour simplifier on pourra se limiter aux entiers naturels uniquement.

Je vous conseille de procéder en 3 étapes :

1. **Étape 1** : écrire un programme qui lit uniquement des entiers passés en paramètres et qui les affiche à l'écran :

   ```
   ./calc 2 4 6 8 10
   2
   4
   6
   8
   10
   ```

2. **Étape 2** : écrire un programme qui lit uniquement des entiers passés en paramètres et les stocke dans une pile LIFO, puis il extrait et affiche les valeurs de la pile :

   ```
   ./calc 2 4 6 8 10
   10
   8
   6
   4
   2
   ```

3. **Étape 3** : implémenter la calculatrice à l'aide de l'algorithme décrit plus haut.

   ```
   ./calc 13 12 + 2 3 + x 2 x
   ```

```
Résultat : 250
./calc 1 2 3 4 + + +
Résultat : 10
./calc 1 2 + 3 + +
Erreur de syntaxe
```

Voilà à vous de jouer ! Mais pour ceux qui auraient un peu de mal, la suite du sujet donne des indications supplémentaires. Les étudiant.e.s à l'aise pourront réaliser les étapes sans lire les indications, mais après chaque étape, lire quand même les indications pour vérifier que vous suivez le bon chemin.

1 Aide pour l'étape 1

On rappelle que si la fonction `main` est implémentée avec le prototype

```
int main(int argc, char** argv)
```

alors `argc` est le nombre d'arguments passés au programme tandis que `argv` est un tableau de chaînes de caractères qui contient chaque argument. Attention l'argument d'indice 0 est en réalité le nom de l'exécutable, pour nous il s'agira donc toujours de `calc`.

Enfin on mentionne l'existence de la fonction `atoi` déclarée dans `stdlib.h` qui permet de convertir une chaîne de caractères en entier.

2 Aide pour l'étape 2

Commencer par implémenter proprement la structure de pile. La méthode la plus simple est d'utiliser un tableau pour stocker les éléments de la pile ainsi que d'un indice pour repérer le sommet de la pile : par exemple il suffit de connaître la taille de la pile. On pourra par exemple utiliser les définitions suivantes :

```
#define MAXSTACK 64

struct stack_s {
    int data[MAXSTACK];
    int size;
};

typedef struct stack_s stack;
```

Ensuite, implémenter chacune des fonctions utiles pour manipuler cette structure de données :

1. Initialisation de la pile
```
void stack_init(stack *s)
```

2. Test du vide
```
bool stack_empty(stack *s)
```

3. Hauteur de la pile
```
int stack_size(stack *s)
```

4. Insertion d'un élément
```
void stack_push(stack *s, int val)
```

5. Extraction d'un élément
```
int stack_pop(stack *s)
```

On mettra en pratique les habitudes de programmation défensive ! vérification des préconditions, cas limites, *etc*

3 Aide pour l'étape 3

Utiliser l'algorithme proposé dans l'énoncé : lire les arguments de la ligne de commande dans l'ordre, procéder à une disinction de cas pour chaque argument pour modifier la pile en conséquence. Manipuler la pile uniquement à l'aide de son interface !

4 Si vous avez fini

- Ajouter des opérations, vous pouvez vous inspirer des boutons de la HP 35.
- Faire la même chose en OCaml