Cours de programmation séquentielle

La calculatrice

1 Préambule

Cette série n'est pas notée. Comme pour le premier semestre, vous pouvez obtenir un bonus de `0.5` sur la note de l'examen en rendant 2/3 des exercices non-notés du semestre. Vous devez mettre le lien de votre repo `git` dans le wiki correspondant sur `Cyberlearn`. **Ne nous rajoutez pas comme reporter!** Vous pouvez à choix laisser le repo public ou alors le mettre en interne. Pour la partie de compréhension écrivez vos réponses dans un fichier `ANSWERS.md` et placez le dans votre repo `git`.

2 Exercices de compréhension

Dessiner ce qui se passe en mémoire dans les bouts de programme suivant en mettant les valeurs de chaque variable et les moment où elles sont modifiées. Il se peut qu'une partie de ces programmes produise des erreurs. Identifiez les et quand cela s'y prête, tentez de corriger le code.

Exercice de compréhension 1 *(Passage par copie, référence)*

```c
void foo(int a) {
    a += 3;
}

void bar(int *a) {
    *a += 3;
}

void baz(int *a) {
    a += 3;
}

int main() {
    int a = 5;
    foo(a);
    bar(&a);
    baz(&a);
}
```

Exercice de compréhension 2 *(`malloc` mon amour)*

```
char *foo(int a) {
    char tab[a];
    for (int i = 0; i < a; ++i) {
        tab[i] = 'a';
    }
    return tab;
}

char *bar(int a) {
    char *tab = malloc(a);
    for (int i = 0; i < a; ++i) {
        tab[i] = 'a';
    }
    return tab;
}

void baz(char *tab, int b) {
    tab = malloc(b);
    for (int i = 0; i < b; ++i) {
        tab[i] = 'a';
    }
}

int main() {
    char *a = foo(4);
    char *b = bar(4);
    char *c = malloc(4);
    baz(c, 4);
    free(a);
    free(b);
    free(c);
}
```

---

3 La calculatrice

3.1 Buts

— Créer et utiliser une librairie de pile générique.
— Implémenter l'algorithme de transformation d'une expression infixe en une expression postfixe.
— Implémenter l'algorithme d'évaluation d'une expression postfixe à l'aide de la pile d'entiers.

Vous avez vu tous ces algorithmes dans le cours d'algorithmique avec le Professeur Albuquerque.

3.2 Enoncé

Il s'agit de réaliser un programme qui, en deux phases, évalue une expression arithmétique composée de nombres réels et des opérations : `+`, `-`, `*`, `/`, `^`.
La première phase transformera l'expression infixe en une expression postfixe. La seconde phase évaluera l'expression postfixe à l'aide d'une pile. Je rappelle ici pour la seconde fois, que ces deux algorithmes ont été vus aux cours du Prof. Dr. Dipl. Paul Albuquerque (voir le lien https://bit.ly/3kYmRex).

3.3 Cahier des charges (pas forcément dans cet ordre)

- Créer une librairie de gestion de pile dynamique de `void *` (avec les fonctionnalités `push`, `pop`, `peek`, `is_empty`). Elle sera utilisée pour la gestion des parenthèses, ainsi que pour l'évaluation de l'expression postfixe.
- Créer une fonction qui prend en argument une expression infixe et qui retourne l'expression postfixe correspondante. Les expressions in/postfixes sont des chaînes de caractères (`char *`).
- Créer une fonction qui prend en argument une expression postfixe, l'évalue à l'aide de la pile dynamique, et retourne un double qui est le résultat de l'évaluation.

Vous devrez probablement implémenter d'autres fonctions, ne vous limitez pas à ce qui est conseillé ici si vous pensez que c'est nécessaire.

3.4 Indications

Inspirez-vous du travail pratique du vecteur générique pour créer votre librairie de pile dynamique. Pour simplifier la lecture de votre chaîne de caractère infixe, nous allons supposer que les opérandes sont des entiers à un seul chiffre de `0` à `9`. Les opérations sont `+`, `-`, `*`, `/`, `^` et que des parenthèses peuvent être présentes. Nous supposerons également qu'il n'y a pas d'espaces dans la chaîne infixe et que finalement il n'y a pas non plus l'opérateur "négatif" (comme dans le nombre entier `-3`).

3.5 Tests

Utilisez la méthode de développement par pilotée par des tests. Ce travail pratique s'y prête parfaitement. Dans votre `Makefile` vous devez créer une cible `tests` qui exécute tous les tests de vos librairies.

Pour contrôler la qualité de votre programme, créez une batterie de tests. Vous devez également créer un programme qui illustre le fonctionnement de votre calculatrice en utilisant les expressions suivantes par exemple:

- `7+2*3=13`
- `8-5-2=1`
- `(7-5)/(2-(12-9)^(2^3))=-0.0003049`
- `21/(7-3*2)=21`
- `3*(5-(7-1)*3)/2=-19.5`
- `5^(3/2)+(5)^(2/3)=14.10`