UNIVERSITÉ DE STRASBOURG

UFR de mathématique et d'informatique Université de Strasbourg

Structures de données et algorithmes 2

Calculatrice polonaise

Yusif IBRAHIMOV

---

1. Sujet

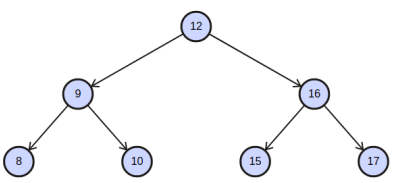
Le but du projet est la mise en œuvre de la calculatrice polonaise.
Il existe 3 types de représentation de l'équation algébrique. Ils sont préfixe, postfixe et infixe.
Infix: <identifier><operator><identifier>
Prefix: <operator><identifier><identifier>
Postfix: <identifier><identifier><operator>
I.e Infix: (a+b)*c Postfix: a b + c * Prefix: * + a b c

La notation préfixe est également appelée notation Polonaise, la notation postfixe est appelée notation Polonaise Inverse. Le but du projet étant la calculatrice polonaise, nous utiliserons la notation de préfixe dans notre projet.

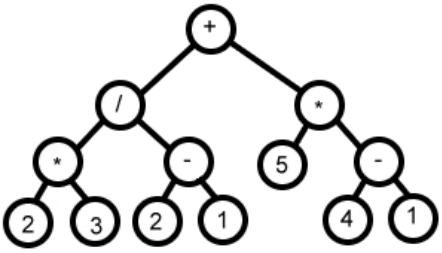
2. Structure de données:

Dans le projet, il nous est demandé d'utiliser la notation d'arbre binaire pour implémenter.
En fait, à partir de SDA1, nous pouvons facilement y penser pour utiliser la structure de données de la pile, mais ici nous utilisons l'arbre binaire (nous l'appellerons alors arbre d'expression binaire)

Voici l'exemple de l'arbre binaire (en fait l'arbre de recherche binaire).
La question est que nous n'avons que les nombres (opérandes) ici, comment saurons-nous où utiliser quel opérateur?
La première idée était de penser au nouvel arbre binaire à propos des opérateurs purement, mais le problème est la fusion de 2 arbres binaires.

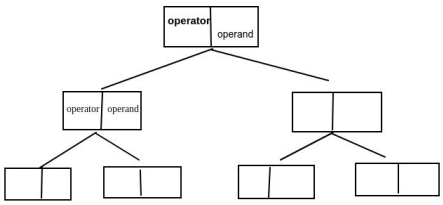


Comme mentionné dans le sujet pdf qui a été donné par le professeur, la meilleure solution est d'utiliser un arbre d'expression binaire qui se soucie à la fois des opérateurs et des opérandes. Voici l'exemple de l'arbre d'expression binaire.
La notation infixe est + * 2/3 - 2 1 * 5 - 4 1



Expression tree for 2*3/(2-1)+5*(4-1)

Comme nous l'avons vu ci-dessus, nous avons 2 types différents dans notre arbre d'expression, l'un est opérateur (chaîne), le second est valeur (double). Comment les combiner? Une façon consiste à utiliser une chaîne pour les deux, puis j'ai décidé de ne pas l'utiliser en raison des erreurs du problème de distinction. Je peux dessiner ici la version solide de mon exemple d'arbre d'expression



l'implémentation de mon arbre est:
Nous avons un type bultin comme "double" mais qu'en est-il de l'opérateur? quoi utiliser pour exprimer l'opérateur. En python, il y a une structure de données comme "dictionnaire", en C nous pouvons implémenter la table de hachage pour l'exprimer. Mais je ne veux pas utiliser la deuxième structure de données, nous pensons aux autres méthos que je vais déclarer ci-dessous:

```c
struct ExpTree {
        double value;
        operation oper;
        struct ExpTree* right, * left;
};
typedef struct ExpTree ExpTree;
```

La méthode la plus rapide consiste à utiliser la technique *d'énumération.*
L'énumération est un type de données défini par l'utilisateur en langage C. Il est utilisé pour attribuer des noms aux constantes intégrales, ce qui facilite la lecture et la maintenance d'un programme. Le mot clé "énumération" est utilisé pour déclarer une énumération. ... Le mot-clé enum est également utilisé pour définir les variables de type enum:

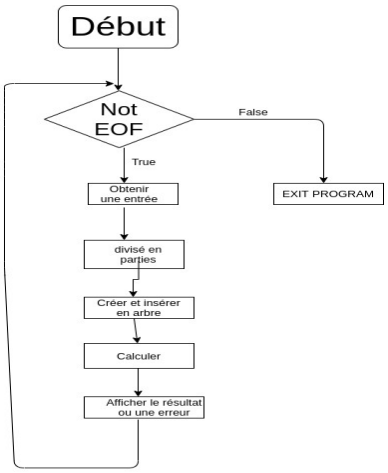
```c
enum operations{
        ADD,SUB,MULT,DIV,MIN,MAX,SQRT,EXP,LN,ABS,FLOOR,CEIL,NOTUP
};
typedef enum operations operation;
```

Ainsi, avec l'aide d'énumération, il y aura un accès facile aux opérateurs:

opération	enum	opération	enum
+	ADD	sqrt	SQRT
-	SUB	exp	EXP
*	MULT	ln	LN
/	DIV	abs	ABS
min	MIN	floor	FLOOR
max	MAX	ceil	CEIL
autre			NOTUP

3. Algorithmes

Notre programme comporte 5 étapes:



1,2 Si ce n'est pas Crld-D envoyé (pas EOF), nous obtenons l'entrée de l'utilisateur sous forme de chaîne. Nous avons 2 fonctions appelées defineOperation () et defineOperand () pour distinguer les opérateurs et l'opérande.

Pour définir si l'opérateur ou non, j'ai utilisé la fonction de hachage de Bernstein. par exemple:

mon inp = "+"
if (hash (inp) == hash ("+)) return ADD
... (jusqu'à CElL)
sinon retourner NOTUP

Remarque: d'abord, j'ai utilisé les fonctions de hachage de Horner, puis j'ai remarqué que la fonction de hachage de Horn donne le même résultat pour "+" et "max"

Notre fonction defineOperand () renvoie un booléen. si la chaîne est incorrecte pour retourner un opérande, elle renvoie la valeur false, sinon elle retourne vrai et modifie un opérande donné en argument

3. Pour minimiser l'utilisation de la fonction de bibliothèque de chaînes, j'ai implémenté certaines d'entre elles. Par exemple: str_len (), str_cpy () et etc.
Supposons que nous ayons un tableau de chaînes. à l'aide de la fonction insert (), nous définissons si l'opérateur est binaire ou unaire, puis nous plaçons les éléments du tableau dans l'arborescence nouvellement créée dans chaque entrée.
à la fin, il retourne un arbre créé.

Mais le sens est que, nous obtenons l'entrée de l'utilisateur sous la forme d'une chaîne unique et non du tableau de la chaîne. pour utiliser la fonction d'insertion, nous devons diviser notre chaîne en différentes parties, puis les mettre dans un tableau. À ce stade, la fonction splitter () nous aide. Il divise la chaîne en parties selon les espaces. Maintenant, nous sommes prêts à créer l'arbre final, en utilisant la fonction expressionTree () qui obtient la chaîne unique et la place dans l'arbre et la renvoie

4. Nous avons une fonction Calculate () qui prend notre arbre en entrée. selon les opérations et operatos de notre arbre il calcule les enfants gauche et droit (si unaire seulement gauche). dans le cas de l'erreur, il renvoie un message d'erreur. (en fait, le type de retour est double, mais je préfère utiliser #define ERRMSG -0.010101 et le renvoyer.

5. C'est la partie de la fonction main(). À l'intérieur de la fonction main, nous obtenons l'entrée de l'utilisateur (stdin). Pour effectuer ce processus jusqu'à l'EOF, nous utilisons des fgets à l'intérieur de la boucle while. Nous avons fait une insensibilité à la casse à l'aide de la fonction makelow () qui modifie notre entrée. nous calculons le résultat à l'aide de la fonction de calcul et l'imprimons avec la notation "\ 033 [1; 34m >>> \ 033 [0m% .4f \ n". Si cette fonction renvoie ERRMSG, nous imprimons "\ 033 [1; 34m >>> \ 033 [0m ERROR \ n".

4. Exécution des codes source et conclusions

<make pc>

```
joseph@joseph-Lenovo-V110-15ISK:~/Desktop/yusif_ibrahimov$ make pc
cc main.o expression.o -o pc -lm
joseph@joseph-Lenovo-V110-15ISK:~/Desktop/yusif_ibrahimov$ ./pc
>>> + 1 2
>>>  3.0000
>>> - * + 1 2 3 4
>>>  5.0000
>>>
```

```
eph@joseph-Lenovo-V110-15ISK:~/Desktop/yusif_ibrahimov$ printf '+ 1 2\n- * 3
n' | ./pc
>>>  3.0000
>>>  7.0000
```

```
joseph@joseph-Lenovo-V110-15ISK:~/Desktop/yusif_ibrahimov$ ./pc
>>> min 3
>>>  ERROR
>>>
```

Les principales difficultés pour moi de choisir la deuxième structure pour les opérateurs. Ensuite, j'ai travaillé avec la structure enum.

Le deuxième problème était la résolution du "défaut de segmentation". Ces erreurs proviennent des pointeurs, j'ai donc dû faire beaucoup de débogages et utiliser le printf () à l'intérieur de chaque bloc.

Le troisième et plus grand problème était la gestion des erreurs. Je ne sais pas où et comment imprimer les rétroviseurs. puis j'ai pensé à la définition ERRMSG.

J'ai eu un tel type de problème qui agissait au maximum comme l'opérateur "+". Ensuite, j'ai remarqué que la fonction de hachage horner renvoie le même index pour "+" et "max". Ensuite, j'ai changé ma fonction de hachage en fonction de hachage Bernstein. Pourquoi j'ai utilisé la fonction Horners pour la première fois, car elle est également célèbre dans les arbres Merkle (blockchain)

Le dernier problème concerne les tests. Il agit de la bonne manière dans les bonnes équations, mais encore une fois, j'ai eu des problèmes de gestion des erreurs, donc mon projet est faisable dans les tests.