ASI3 Mardi 15 Janvier 2019

Algorithmique et Bases de la programmation

Durée : *1h55*
Documents autorisés : **AUCUN**

L'objectif de cet examen pratique est de développer la calculatrice dont l'analyse descendante et les algorithmes ont été vu en TD et en examen théorique.

Informations

Avant de commencer l'examen

Connectez vous à la machine et suivez les instructions suivantes :

— copiez l'archive `C.tar.gz` qui se trouve dans `/opt/files` dans votre répertoire (commande `cp`) ;
— après l'avoir décompressée (commande `tar` avec les options `zxvf`), renommez le répertoire en lui donnant comme nom : 'EXAM-ALGO-' (les tirets sont ceux du signe moins) suivi de votre nom et prénom, sans espace, sans caractères accentués et en minuscule sauf pour les premières lettres du prénom et du nom. Par exemple, si « Paul Du Villaré » devait passer cet examen, il nommerai son répertoire EXAM-ALGO-DuvillarePaul.

À la fin de l'examen

Cinq minutes avant la fin de l'examen :

— assurez vous que les fichiers sont bien au format ISO-8859 (c'est par défaut le cas) ;
— **assurez vous qu'il n'y a aucun accent dans les commentaires que vous auriez pu ajouter ;**
— assurez vous que votre projet compile ;
— créez une archive de votre répertoire (commande `tar` avec les options `zcvf`), que vous nommerez comme le répertoire (avec le préfixe `.tar.gz`) ;
— déposez sur moodle votre archive (cours « algorithmique avancée et programmation C », section « examens pratiques »).

Attention :

— le lien de dépôt est actif uniquement 10 minutes, si vous dépassez le délai, votre note sera 0 ;
— on ne peut déposer qu'une seule fois le projet sur moodle.

Quelques conseils

Pour réussir votre examen suivez les conseils suivants :

— respectez bien les consignes données ci dessus. **Le fait de ne pas les respecter vous fera perdre 2 points sur la note finale** ;

— par défaut le projet compile et s'exécute. Après le développement de chaque fonction C, compilez votre projet (`make`) et ne passez à la fonction suivante que lorsque votre projet compile : **le fait de rendre un projet qui ne compile pas ou ne s'exécute pas (par exemple à cause d'un *segmentation fault*), divisera par deux votre note finale** ;
— la note finale sera fonction :
  — du nombre de tests unitaires qui seront valides ;
  — du nombre de *Warning* (compilation avec l'option `-Wall`) qui seront affichés ;
  — de la quantité, de la qualité et de la lisibilité du code (indentation et nom des identifiants).

1 Calculatrice

L'objectif de cet examen pratique est de développer un programme C qui évalue une expression arithmétique représentée par une chaîne de caractères. On obtiendra alors un programme dont l'exécution pourra ressembler à :

```
$ bin/calc "2,5*((3.5*.25)+(25.-17))"
22.187500
```

1.1 Analyse

Nous avons précédemment étudié une analyse descendante permettant de calculer une expression arithmétique (contenant uniquement des nombres positifs) représentée par une chaîne de caractères. La figure 1 présente cette analyse.

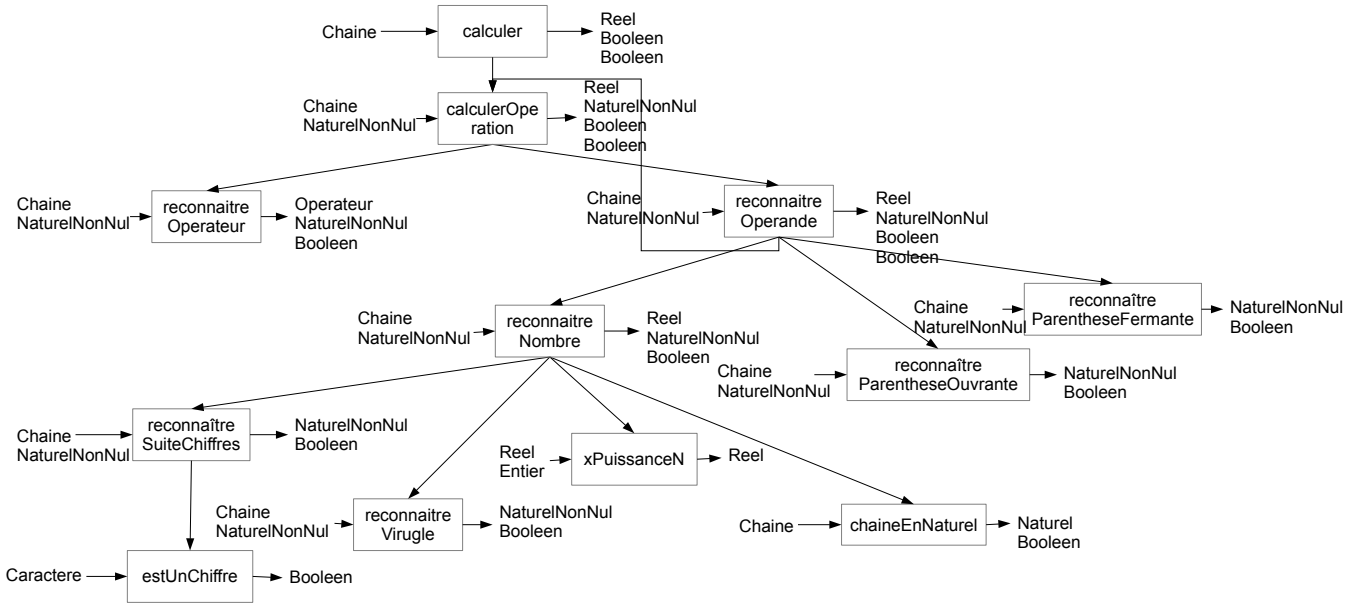

FIGURE 1 – Analyse descendante

Pour rappel :
— la chaîne de caractères en entrée de chaque opération représente la chaîne dans sa globalité ;
— le naturel non nul en entrée de chaque opération représente l'indice de la chaîne où débute de l'analyse effectuée par l'opération ;

— le premier booléen en sortie de chaque opération permet de savoir si la chaîne en entrée représente bien syntaxiquement une expression arithmétique ;
— le deuxième booléen en sortie de certaines opérations permet de savoir si une erreur sémantique s'est produite ou pas ;
— le naturel non nul en sortie de chaque opération représente l'indice de la chaîne où se termine l'analyse effectuée par l'opération dans le cas ou le premier booléen précédent vaut VRAI. Si ce dernier vaut FAUX, ce naturel non nul vaut la même valeur que celle fournie en entrée.

Les contraintes d'utilisation du programme sont :
— la chaîne doit correspondre à une opération arithmétique simple : opérande opérateur opérande ;
— une opérande est un nombre (entier ou réel) positif ou une opération arithmétique entre parenthèse ;
— le caractère représentant le virgule des nombres réels peut être '.' ou ',' ;
— le caractère représentant les parenthèses sont uniquement '(' pour la parenthèse ouvrante et ')' pour la parenthèse fermante ;
— les opérations sont uniqument l'addition (caractère '+'), la soustraction (caractère '-'), la multiplication (caractère '*') et la division (caractère '/') ;
— lorsque la chaîne n'est syntaxiquement pas une expression arithmétique le programme affiche `"Erreur de syntaxe"` sur la sortie standard ;
— lorsque l'expression arithmétique est sémantiquement fausse, le programme affiche `"Erreur de semantique"` sur la sortie standard.

1.2 Conception

Les différents algorithmes des fonctions et procédures correspondant aux opérations de l'analyse descendante ont été vus en TD. Seule la procédure `reconnaitreOperande` a été complétée de façon à prendre en compte le fait qu'une opérande puisse être une opération. Voici son algorithme :

procédure reconnaitreOperande **(E** leTexte : **Chaine de caracteres**, debut : **Naturel** ; **S** leReel : **Reel**, prochainDebut : **NaturelNonNul**, syntaxiquementOK, semantiquementOK booleen**)**

Déclaration debutOperation, debutParentheseFermente : **NaturelNonNul**

debut
semantiquementOK ← VRAI
reconnaitreNombre(leTexte,debut,leReel,prochainDebut,syntaxiquementOK)
si non syntaxiquementOK **alors**
reconnaitreParentheseOuvrante(leTexte,debut,prochainDebut,syntaxiquementOK)
si syntaxiquementOK **alors**
debutOperation ← prochainDebut
calculerOperation(leTexte,debutOperation,leReel,syntaxiquementOK,
semantiquementOK,prochainDebut)
si syntaxiquementOK et semantiquementOK **alors**
debutParentheseFermente ← prochainDebut
reconnaitreParentheseFermente(leTexte,debut,prochainDebut,syntaxiquementOK)
si non syntaxiquementOK **alors**
prochainDebut ← debut
finsi
sinon
prochainDebut ← debut
finsi
sinon

```
        prochainDebut ← debut
      finsi
    finsi
fin
```

1.3 Le programme

Le programme est composé des fichiers suivants :

— `include/stringext.h` qui déclare la fonction `strsubstring` qui permet de récupérer la sous-chaîne d'un chaîne (à l'image des fonctions proposées par `string.h`, l'allocation de l'espace permettant de stocker la chaîne demandée est à la charge de l'utilisateur) ;
— `include/calc.h` qui déclare la signature de la fonction C `CALC_calculer` ;
— `include/calcPrive.h` qui déclare les signatures des fonctions de l'analyse descendante, ainsi que les types et constantes privées ;
— `src/stringext.c` qui définit de la fonction `strsubstring` ;
— `src/calc.c` qui définit les fonctions déclarées dans `include/calc.h` et `include/calcPrive.h` ;
— `src/calcTU.c` le programme des tests unitaires des fonctions de `calc.c` ;
— `src/main.c` le programme principal.

L'exécution du `make` générera le programme principal `bin/calc` et le programme des tests unitaires `test/calcTU`.

2 Travail à réaliser

Il vous faut compléter le fichier `calc.c` de façon à ce que le maximum de tests unitaires fonctionnent (`test/calcTU`). Si tous ces tests fonctionnent, le programme devrait fonctionner...