INF404 - Travaux pratiques - Séance 2 et/ou 3

Evaluation d'expressions arithmétiques entièrement parenthésées

Avant de commencer cette séance :

1. Créez un répertoire ***TP2*** dans votre répertoire ***INF404***
2. Placez-vous dans ***INF404/TP2*** et recopiez les fichiers utilisés pendant le TP1 : `cp ../TP1/* .`

Objectifs

L'objectif de cette séance est de programmer l'évaluation des expressions arithmétiques entièrement parenthésées (eaep) vues en TD. On rappelle la **grammaire** définissant la syntaxe de ces expressions [1] :

op	→	`PLUS`
op	→	`MOINS`
op	→	`MUL`
eaep	→	`ENTIER`
eaep	→	`PARENTH_OUVRANTE` eaep `op` eaep `PARENTH_FERMANTE`

Exercice 1 - étendre l'analyse lexicale

Modifiez les fichiers ***analyse_lexicale.c*** et ***analyse_lexicale.h*** pour que votre analyseur lexical reconnaisse les lexèmes `PARENTH_OUVRANTE` et `PARENTH_FERMANTE`.
Compilez le fichier `test_lexeme` : `make test_lexeme`
Ecrivez (avec un éditeur de votre choix) un fichier ***entree.txt*** contenant une séquence de lexème "correcte" (sur une seule ligne).
Exécutez ***test_lexeme*** sur cette séquence : `./test_lexeme entree.txt`
Vérifiez que les erreurs lexicales sont bien detectées ...

Exercice 2 - analyse syntaxique d'une EAEP

Modifiez maintenant la fonction ***analyse_syntaxique*** pour fournir l'interface ci-dessous. Vous **utiliserez pour cela des procédures `Rec_eaep` et `Rec_op` comme vu en cours**
(c.f. transparents 11 du fichier :
`http://www-verimag.imag.fr/~mounier/Enseignement/INF404/slides_cours_3.pdf`)

1. pour rester compatible avec l'analyseur lexical la convention adoptée ici est l'inverse de celle du cours : les symboles terminaux (i.e., les lexèmes) sont en **majuscules** et les non-terminaux en **minuscules**

```
void analyser (char *nom_fichier) ;
   -- e.i : indifferent
   -- e.f : une EAEP a ete lue dans le fichier de nom nom_fichier
   -- si elle ne contient pas d'erreur un message est affiche
   -- sinon une Erreur_Syntaxique est signalee
```

Ecrivez un programme principal qui appelle cette procédure et testez ce programme sur divers exemples.

Exercice 3 - évaluer une expression sur un arbre abstrait

Copiez dans votre répertoire les fichiers suivants depuis le répertoire `/Public/404_INF_Public/TPs/TP2` :

- **type_ast.h**, spécification d'un arbre abstrait ;
- **ast_construction.h** et **ast_construction.c**, spécification et corps des primitives de construction d'un arbre abstrait ;
- **ast_parcours.h**, **ast_parcours.c**, spécification et corps des primitives de parcours d'un arbre abstrait ;
- **essai_ast.c**, programme principal (de test).

Examinez ces fichiers ...

Complétez le corps de la fonction `evaluation` du fichier **ast_parcours.c** contenant le corps des fonctions définies dans **ast_parcours.h**.

Compilez le programme principal **essai_ast.c**, et exécutez-le. Vérifiez qu'il affiche bien l'expression arithmétique correspondant à l'arbre défini dans la variable `expression`.

Testez ensuite votre programme sur les expressions suivantes (en modifiant **essai_ast.c** pour construire les arbre abstraits correspondants) :

- `((2 + 3) * (5 - 2))`
- `((2 + 3) / (5 - 2))`
- `(3 / ((5 - 2) - 3))`

Exercice 4 - construction de l'arbre abstrait

Complétez maintenant l'analyse syntaxique écrite dans l'exercice 2 afin de produire un **arbre abstrait** de l'expression analysée. Vous pouvez vous aider des transparents 12 et 13 en ligne à l'adresse suivante : `http://www-verimag.imag.fr/~mounier/Enseignement/INF404/slides_cours_3.pdf`

Modifiez ensuite votre programme principal pour :

1. analyser et produire l'arbre abstrait d'une expression *eaep* lue au clavier
2. afficher cet arbre abstrait (fonction `afficher` de `ast_parcours.c`)
3. calculer et afficher la valeur de l'expression (fonction `evaluer` de `ast_parcours.c`)