TP n°3, I41 – Projet *Emmanuel Bruno*
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Le module d'analyse syntaxique

Objectif

Réaliser la base du module d'analyse syntaxique. Ce module s'appuie sur le module d'analyse lexicale pour extraire à la demande les symboles d'un programme. Il permettra notamment de dire si un programme est bien conforme à la syntaxe et, le cas échéant, d'indiquer d'éventuelles erreurs.

ATTENTION : comme pour le TP sur l'analyse lexicale, la structure du programme est imposée et une partie du code est fournie ; copier les fichiers `algo_syntaxique.c` et `algo_syntaxique.h` .

Ce programme dépend des fonctions définies lors du TP précédent. Les fichiers `algo_lexical.c` et `algo_lexical.h`, devront être transformés en un fichier algo_lexical.o.
algo_lexical.h devra être nclus dans le fichier `algo_syntaxique.c`.

REMARQUE : dans chaque procédure, vous **DEVREZ** afficher ce que vous lisez au fur et à mesure. Par exemple, dans la procédure facteur afficher dès le début "DF" (début de facteur), puis quand vous lisez un identificateur, un nombre, ... affichez sa valeur (ex: "identificateur : A"), finalement à la fin de la procédure afficher "FF" (fin de facteur). Faites de même pour toutes les procédures (DT début de terme, FT fin de terme, ...).
Lors de l'éxécution du programme l'imbrication des DF, FF, DT, FT, ... vous indiquera que votre expression est bien analysée.

1. Termes des expressions

1.1. Réalisation

Compléter les procédures `facteur` et `terme` comme cela a été vu en TD. Cela doit permettre d'analyser des expressions du style `titi * b div 12`.

1.2. Test

Créer un fichier texte `terme.txt` puis exécuter la procédure `exercice1`.
Faites la liste des cas qui posent problèmes et restez ces différents termes.

2. Expressions simples

2.1. Réalisation

Compléter la procédure `expression-simple` qui permet de gérer les opérateurs moins prioritaires (addition, …) en procédant comme précédemment (cf. grammaire du langage et TD). On peut alors analyser des expressions de la forme `32 * titi + 3 mod 2 - 45.5`.

2.2. Test

Créer un fichier texte `expression-simple.txt` puis exécuter la procédure `exercice2`. Faites la liste des cas qui posent problèmes et restez ces différents termes.

3. Expressions

3.1. Réalisation

Compléter la procédure `expression` pour gérer convenablement les opérateurs de comparaison. On peut donc maintenant analyser des expressions de la forme `(titi >= toto) ou (reste > 0)`.

3.2. Test

Créer un fichier texte `expression.txt` puis exécuter la procédure `exercice3`. Tester différentes expressions.

Pour aller plus loin (cette partie est facultative)

4. Listes d'instructions

4.1. Réalisation

On peut maintenant généraliser la méthode à l'analyse des instructions du langage. Créer la procédure `instruction` qui selont le mot reservé qu'elles trouvent execute l'un des procédure suivantre :
`montrer_la_valeur_de`,`demander_une_valeur_pour`, `affecter_la_valeur_de`, `si_alors`, `tant_que`

4.2. Test

Créer un fichier texte `instruction.txt` puis exécuter la procédure `exercice4`. Tester différentes instructions.

5. Algorithme complet

5.1. Réalisation

De même, on peut poser la dernière pièce en analysant un algorithme en général (procédure `algorithme` à compléter), tout d'abord **sans gérer** les déclarations.
Puis, on peut analyser les déclarations (procédures `declarations`, `suite_declaration` et `liste_variables`) et les intégrer à `algorithme`, pour cela attendez la partie facultative qui propose la gestion des variables.

5.2. Test

Créer un fichier texte `algo.txt` puis exécuter la procédure `exercice5`. Tester différents algorithmes.