INF404 - Travaux pratiques - Séance 5 et/ou 6

Séquence d'affectations

Avant de commencer cette séance :

1. Créez un répertoire ***TP5*** dans votre répertoire ***INF404***
2. Placez-vous dans ***INF404/TP5*** et recopiez les fichiers utilisés pendant le TP4 : `cp ../TP4/* .`
3. aidez-vous des **transparents du cours 6** le cas échéant ...

Et en fin de séance transmettez votre **compte-rendu** dans la zone de dépôt prévue sur Moodle !

L'objectif de ce TP est de réaliser un **interpreteur** pour des programmes constitués de **séquences d'affectations**. Avant de passer à un langage plus "complet" cette phase intermédiaire permettra :

— de mieux cerner le comportement attendu d'un interpréteur ;
— d'enrichir le langage des expressions avec les notions (indispensables pour la suite !) d'**affectation** et d'**identificateur**.

Concrètement, l'interpréteur à réaliser doit pouvoir au minimum :

— analyser (analyse lexicale et syntaxique) une séquence d'affectation ;
— afficher la valeur des variables, soit après chaque affectation, soit à la fin du programme.

La définition du langage (lexique et syntaxe) est libre, on donne ci-dessous un exemple :

```
x = 5 ;
y = x + 1 ; y = y * 2 + x ;
toto = x + y ;
```

Etape 1 - définition du lexique

Il s'agit tout d'abord de choisir commment écrire les nouveaux éléments lexicaux :

— les identificateurs (`IDF`) : ce peut être au choix une lettre **unique** ou une **séquence** de lettres (majuscules ou minuscules) ; on peut également y ajouter d'autres caractères. Il est toutefois préférable que le 1er caractère d'un identificateur ne soit pas un chiffre ...
— le symbole d'affectation (`AFF`) : on peut choisir le caractère "`=`", ou un "`:=`", ou "`<-`", ou autre ...
— le symbole séparant deux affectations (`SEPAFF`) : ce peut être un point-virgule, ou une fin de ligne, ou autre. On peut aussi choisir de considérer la fin de ligne comme un séparateur (au même titre que l'espace).

Une fois ces choix effectués modifiez le module d'analyse lexicale pour intégrer ces modifications. **Testez le résultat** avec le programme **test_lexeme** avant d'aller plus loin ... !

Etape 2 - définition de la syntaxe d'une affectation

On doit maintenant préciser la syntaxe du langage en intégrant ces nouveaux lexèmes. On peut par exemple commpléter la grammaire des Eag du TP3 en **ajoutant** les deux règles suivantes :

$$\begin{aligned} \text{aff} &\rightarrow \texttt{IDF AFF} \text{ eag } \texttt{SEPAFF} \\ \text{facteur} &\rightarrow \texttt{IDF} \end{aligned}$$

L'axiome `aff` définit une affectation. On peut modifier la procédure `rec_facteur` et ajouter une procédure `rec_affectation` dans l'analyse syntaxique pour prendre en compte ces modifications (en laissant la construction de l'Ast de coté pour le moment ...).

Etape 3 - Interpréter une affectation et table des symboles

Une solution possible pour *interpréter* une affectation de la forme `x = exp` consiste à procéder comme suit :

- Chaque identificateur est mémorisé dans une structure qui contient son *nom* et sa *valeur* courante. On appelera cette structure *table des symboles*.
- Lorsque l'on rencontre un identificateur dans une la partie droite (`exp`) d'une affectation celui-ci doit se trouver dans la table des symboles (il a du être en partie gauche d'une affectation précédente). On connait donc sa valeur, on peut créer le noeud de l'Ast correspondant ;
- Lorsque l'Ast de la partie droite d'une affectation est entièrement construit, on l'évalue ;
- L'identificateur correspondant à la partie gauche (`x`) est alors inséré dans la table des symboles (**s'il n'y est pas déjà !**) avec son nom et sa valeur (si cet identificateur était déjà présent dans la table son ancienne valeur est mise à jour).

Ecrivez un module de gestion de table des symboles qui implémente les primitives nécessaires (initialiser une table vide, inserer un identificateur, consulter/modifier sa valeur, imprimer l'ensemble de la table des symboles). Vous avez le choix de la structure de données, un **tableau de taille fixe** peut faire l'affaire, mais vous pouvez aussi utiliser une **liste chaînée** si vous préférez. Un exemple de fichier `table_symbole.h` (avec tableau de taille fixe) est donné sur Moodle, il vous suffit donc d'écrire le fichier `table_symbole.c` correspondant.

Etape 4 - Modifier l'analyse syntaxique

Il s'agit maintenant de modifier à nouveau l'analyse syntaxique pour interpréter une affectation : création de l'Ast de la partie droite, évaluation, mise à jour de la table des symboles en insérant l'identificateur de la partie droite (ou en modifiant sa valeur s'il était déjà présent). Toutes ces modifications devraient concerner uniquement les procédures `rec_facteur` et `rec_affectation` ...

On peut alors tester le fonctionnement de l'interpréteur sur une seule affectation en écrivant un programme principal **interpreteur** qui analyse **une** affectation (en appelant `rec_affectation`) et qui affiche la table des symboles.

Etape 5 - Généraliser à plusieurs affectations

Pour généraliser à plusieurs affectations il suffit de modifier la grammaire (et l'analyse syntaxique !) en ajoutant les règles suivantes :

$$\begin{aligned} \text{seq\_aff} &\rightarrow \text{aff seq\_aff} \\ \text{seq\_aff} &\rightarrow \texttt{FIN\_SEQUENCE} \end{aligned}$$

Variantes et extensions ...

Quelques pistes pour prolonger cette version "minimale" de l'interpréteur, et anticiper sur la suite :

- imposer que les variables soit déclarées en début de programme (et donc détecter les doubles déclarations, les variables non déclarées) ;

— paramètrer l'interpréteur pour que l'utilisateur puisse choisir quelles variables il veut afficher, à quelles instructions ? Prévoir un mode "exécution en pas à pas" ;
— ajouter des instructions `read(x)` et `print(x)`, qui permettent de lire et écrire une valeur au clavier.