INF404 - Travaux pratiques - Séances 6 et/ou 7

Interpréteur : instruction conditionnelle

Ce TP est à adapter en fonction du langage que vous avez choisi de traiter dans le projet ...

Avant de commencer cette séance :

1. Créez un répertoire ***TP7*** dans votre répertoire ***INF404***
2. Placez-vous dans ***INF404/TP7*** et recopiez les fichiers utilisés pendant le TP6 : `cp ../TP6/* .`

Il est **fortement conseillé** de bien relire les **transparents du cours 8** avant de commencer ce TP !

L'objectif de ce TP est de compléter l'**interpreteur** commencé lors du précédent TP. On rappelle que cet interpréteur 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) reste libre, on donne ci-dessous un exemple :

```
x = 5 ; y = 1 ;
if  x 10 then
    x=x+1
else
   y =y*2 ;
fi ;
ecrire (y) ;
```

Lexique et Syntaxe du langage

Dans ce langage un programme est une suite (non vide) d'instructions. Une instruction peut être soit une affectation, soit une instruction conditionnele (`if`), soit une instruction d'entrée-sortie (`lire`, `ecrire`), etc.

La grammaire de ce langage pourra donc être de la forme :

pgm	$\rightarrow$	seq_inst
seq_inst	$\rightarrow$	inst suite_seq_inst
suite_seq_inst	$\rightarrow$	`SEPINST` seq_inst
seq_inst	$\rightarrow$	ε
inst	$\rightarrow$	`IDF AFF` eag
inst	$\rightarrow$	autres formes d'instructions ...

On pourra choisir de représenter le lexeme `SEPINST` par un `';'`, et on complètera le lexique au fur et à mesure de l'ajout de nouvelles instructions ....

T. SVP

Instruction conditionnelle

On donne ci-dessous un exemple de règle pour l'instruction conditionnelle :

$$\begin{aligned} \text{inst} &\rightarrow \texttt{IF}\ \text{condition}\ \texttt{THEN}\ \text{seq\_inst}\ \texttt{ELSE}\ \text{seq\_inst}\ \texttt{FI} \\ \text{condition} &\rightarrow \text{eag}\ \texttt{OPCOMP}\ \text{eag} \end{aligned}$$

Ici "condition" désigne une expression booléenne avec une syntaxe simple dans laquelle `OPCOMP` désigne un opérateur de comparaison ($=$, $\neq$, $\leq$, $\geq$, $<$, $>$, etc.) entre deux expressions arithmétiques.

Compléter l'analyse lexicale et syntaxique de votre interpréteur pour prendre en compte cette instruction conditionnelle (sans construction de l'AsT). N'oubliez pas de vérifier que votre analyse lexicale fonctionne (avec le programme `test_lexeme`) avant de coder l'analyse syntaxique ...

Interprétation

L'étape d'interprétation consiste à exécuter le programme fourni en entrée. Il s'agit donc ici de modifier à nouveau l'analyse syntaxique pour construire un arbre abstrait complet de l'ensemble du programme lu. Il faut donc définir cet arbre abstrait (quelles infos mémoriser pour chaque instruction, comment représenter une séquence d'instruction ?). Il faudra ensuite compléter la définition du type Âst en fonction de ces choix.

Vous pourrez alors écrire ensuite une nouvelle fonction de parcours de l'arbre abstrait complet du programme permettant de calculer (dans la table des symboles) la valeur des variables à chaque instruction ...

Utilisez les algorithmes fournis sur les **transparents du cours 8** pour implémenter cette étape.

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" ;
- réfléchir à l'ajout d'une instruction itérative (`while` ...) !