UEF4.3. Programmation Orientée Objet

Travail Pratique

Très important

- La durée de cette séance est de 2h30.
- Lisez attentivement le sujet. Il est interdit de poser des questions avant 30 min de lecture.
- A la fin de la séance, vous devrez remettre un rapport expliquant votre conception.
- Votre rapport ne doit pas contenir du code source. Seule la conception est demandée.
- La propreté et la lisibilité du rapport seront prises en considération.

Un mini interpréteur

Lorsqu'un compilateur lit le code source d'un programme écrit en java par exemple, il rencontre des noms de variables, de méthodes et de classes. Ces noms, appelés symboles, doivent être mémorisés par le compilateur afin de les reconnaître s'il les rencontre plus loin dans le programme. Pour cela, le compilateur utilise une table appelée *table des symboles* dans laquelle chaque symbole est stocké avec les informations qui lui sont associées.

Pour ce TP, nous allons considérer un compilateur très simple qui peut compiler et exécuter un programme écrit dans un pseudo langage de programmation imaginé pour les besoins du TP. Notre compilateur analysera et exécutera le programme ligne par ligne et c'est la raison pour laquelle nous l'appelons *interpréteur*.

Présentation du pseudo-langage de programmation

Un programme peut contenir des variables (le nom d'une variable est une chaîne de caractères alphanumérique qui doit commencer par une lettre), des expressions mathématiques simples et des commandes. On supposera que les expressions mathématiques peuvent contenir des variables, des opérateurs (+, -, *, /, ^ (opérateur de puissance)), des constantes (nombres réels), des parenthèses ainsi que des fonctions mathématiques standards : sin, cos, tan, abs, sqrt et log.

Notre pseudo langage de programmation peut comprendre deux types de commandes seulement : **let** et **print**.

La commande **let** sert à affecter une valeur à une variable. Exemples :

```
let x = 5
let x = 2+3
```

La commande **print** sert à afficher la valeur d'une variable ou le résultat d'une expression. Exemples :

```
print  x
print  sin(3*x-7)+log(sqrt(y))
```

La figure 1 montre la définition d'une ligne de commande suivant la forme BNF présentée en Annexe 1.

```
<ligne-de-commande>   ::=  "print" <expression> |"let" <variable> "=" <expression>

<expression>          ::=  [ "-" ] <term> [ [ "+" | "-" ] <term> ]...

<terme>               ::=  <facteur> [ [ "*" | "/" ] <facteur> ]...

<facteur>             ::=  <element> [ "^" <element> ]...

<element>             ::=  <nombre> | <variable> | "(" <expression> ")"

                           | <nom-fonction-standard> "(" <expression> ")"

<variable>            ::=  <lettre> { <lettre> | <digit> }

<nombre>              ::=  <digit> {<digit>}
```

Figure 1. Définition d'une commande suivant la forme BNF

Lorsque la commande **let** est exécutée, l'ordinateur doit sauvegarder la variable et les informations relatives à celle-ci dans la table des symboles. A l'exécution d'une commande **print**, l'ordinateur évalue l'expression et affiche sa valeur à l'écran. Si l'expression contient une variable, le compilateur doit récupérer la valeur de cette variable à partir de la table des symboles s'il l'y trouve, ou déclarer une erreur dans le cas contraire.

La table des symboles est une structure de données comportant une entrée pour chaque symbole (variable ou fonction standard) donnant accès à des champs conservant ses informations. Pour ce TP, nous sauvegardons pour chaque variable, son nom et sa valeur (nous supposons que toutes les variables sont de type double). Pour chaque fonction standard, nous sauvegardons son nom et son code indiquant la fonction à appliquer.

A la rencontre d'un mot, le compilateur doit être capable de distinguer si c'est une commande, le nom d'une variable ou le nom d'une fonction. L'utilisateur n'a donc pas le droit d'utiliser des noms de commandes ou de fonctions standards comme noms de variables.

Fonctionnement de l'interpréteur :

L'utilisateur entre son programme ligne par ligne (une ligne est appelée ligne de commande). Une ligne de commande ne peut pas contenir plus d'une commande, et ne peut pas être écrite sur plus d'une ligne. Lorsque l'utilisateur tape sur la touche entrée du clavier, la ligne de commande saisie est analysée par l'interpréteur. Si celle-ci ne comporte aucune erreur, l'interpréteur renvoie le résultat de l'exécution. Celui-ci peut être :

- L'affichage de « ok » si l'instruction est une commande **let**

- L'affichage de la valeur d'une variable ou d'une expression mathématique dans le cas d'une commande **print**

Dans le cas où la ligne de commande comporte une erreur, un message d'erreur est affiché à l'utilisateur précisant le type de l'erreur commise. Des exemples d'exécution de commandes sont présentés en Annexe 2.

Travail demandé

Concevoir et réaliser le mini interpréteur présenté dans ce TP, suivant l'approche orientée objet

a. Tracez le diagramme des classes en précisant le type des classes (classe abstraite, interface, énumération) et les liens entre elles (héritage, implémentation, utilisation).

b. Précisez pour chacune des classes ses attributs et ses méthodes (hormis les constructeurs, les getters et les setters) en utilisant le tableau suivant.

Type-classe nom-classe	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
...	…
Liste des méthodes	
Modificateur d'accès Valeur de retour Signature de la méthode	Rôle
…	…

Annexe 1 : La notation BNF

La forme de Backus-Naur (souvent abrégée en BNF, de l'anglais (John) Backus (Peter) Naur *Form*) est une notation permettant de décrire les règles syntaxiques des langages de programmation. Elle a été introduite pour décrire le langage de programmation ALGOL60 à sa création et a été, depuis, très utilisée pour présenter la syntaxe d'un nouveau langage. Les meta-symboles de BNF sont :

Meta-symbole	Signification
::=	"est défini comme"
|	"ou"
< >	Ces signes encadrent un nom de catégorie ("symboles non-terminaux" par oppositions aux terminaux représentés entre cotes).
[ ]	Articles facultatifs (optionnels)
{ }	Les accolades encadrent des articles répétitifs
" "	Les cotes encadrent les terminaux (constantes)

Annexe 2 : exemple d'exécution d'un programme

```
Entrez vos commandes. Tapez end pour terminer votre programme.

Une commande doit être de la forme

let <variable> = <expression>
ou
print <expression>


>   let x = 2
Ok

>   let y = -5
Ok

>   print sin(x+ abs(y))*sqrt(9)

La valeur est : 0.365608030215442

> print z

Erreur : variable z non déclarée

> let z = 7*
Erreur : Expression erronée

> print x*(y+4)-(log(10)

Erreur : parenthèse fermante manquante

> end
Fin du programme
```