

Interpréteur de langage LIR

Projet proposé par Frédéric Barrios pour des groupes de 4 personnes

Ce sujet vous propose d'écrire un interpréteur d'un langage de programmation simple.
Cet interpréteur sera nommé **LIR** (pour Langage IUT de Rodez) dans la suite du sujet.

Un interpréteur est un automate enchaînant les tâches suivantes : analyse lexico-syntaxique d'une ligne de commande puis interprétation.

Une ligne entrée par un utilisateur sera donc : soit une commande à exécuter immédiatement, soit une ligne de programme à mémoriser pour une exécution ultérieure. Une ligne de programme se distinguera d'une ligne de commande par le fait qu'elle sera toujours précédée d'un "numéro d'ordre" appelé aussi « étiquette ».

Syntaxe d'une ligne de programme :

numéro **instruction** argument(s)

Exemple : 10 **var** a=110

Syntaxe d'une commande :

commande argument(s)

Exemple : **sauve** nomfichier

Les instructions et les commandes sont des mots clés du langage.

1 LES ELEMENTS DU LANGAGE

1.1 **Mots clés**

Dans la suite les mots clés du langage LIR, c'est-à-dire les instructions et les commandes, seront présentés en **gras**.

Leur liste est la suivante par ordre lexicographique :

affiche charge debut defs efface entre fin lance liste procedure retour sauve si stop vaen var

Ils ne peuvent pas être redéfinis par le programmeur et ne peuvent donc pas servir en tant qu'identificateurs.

1.2 *Etiquettes*

Les étiquettes sont des numéros servant à repérer les lignes d'un programme et à les mettre en ordre. Ces numéros sont des entiers positifs entre 1 et 99999.

L'usage est en général de numéroté initialement les lignes de 5 en 5, ou bien de 10 en 10, afin de laisser de la place pour une insertion éventuelle.

Dans la suite les étiquettes seront désignées par <etiquette>

1.3 *Constantes littérales*

Elles seront de 2 types : entier signé ou chaînes de caractères.

Les valeurs entières sont exprimées en base 10 et donc formées des caractères +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Leur valeur sera comprise dans l'intervalle $[-2^{31} = -2147483648, 2^{31} - 1 = 2147483647]$

Exemples : 12 -243 +15

Les chaînes seront entourées de guillemets et limitées à 70 caractères.

Exemple : "Ceci est une chaîne"

1.4 *Identificateurs*

Les noms identifiant des entiers commenceront obligatoirement par une lettre qui sera suivie d'au plus 24 lettres ou chiffres.

Exemples : `alpha` `indice1` sont des identificateurs corrects pour des variables entières

Les noms identifiant des chaînes de caractères commenceront par le symbole \$, suivi d'une lettre, qui sera suivie d'au plus 24 lettres ou chiffres.

Exemples : `$prenom` `$ligne0` sont des identificateurs de chaîne corrects

Les identificateurs n'ont pas besoin d'être déclarés : leur première utilisation suffit à les créer.

Si cette première utilisation n'est pas une affectation ou une entrée leur valeur sera choisie arbitrairement et cela conduira à un comportement aléatoire...

=> il incombe donc au programmeur de bien maîtriser les initialisations.

Toutes les données utilisées seront variables. Leurs valeurs sont conservées et modifiables durant la session de l'interpréteur. Elles ne sont détruites que par la commande **debut**.

La portée des variables est donc globale (aucun principe de localité).

Leur type est choisi lors de leur première utilisation et il ne peut changer (pas de transtypage)

=> si une instruction suivante dénote une incompatibilité de type alors une erreur intervient (pas de conversion implicite)

Dans la suite un identificateur sera repéré par <identificateur>

1.5 Expressions arithmétiques

Syntaxe d'une expression : opérande1 opérateur opérande2

Les expressions concerneront donc toujours deux opérandes séparés par un opérateur (notation infixe). La présence d'espace (séparateur neutre) n'est pas obligatoire (mais conseillée)

Un opérande est soit une constante littérale, soit un identificateur.

L'opérateur est un caractère symbolisant les opérations arithmétiques courantes : + - * / %

- + = addition entière
- = soustraction entière
- * = multiplication entière
- / = quotient de la division entière
- % = reste de la division entière

Exemples : n + p a /100 a*a

1.6 Expressions sur chaînes

La seule opération utilisée sur les chaînes est la concaténation.

Syntaxe : chaîne1+chaîne2

où chaîne1 et chaîne2 sont des opérandes chaînes c'est-à-dire constantes chaînes ou identificateurs commençant par \$.

1.7 Expressions logiques

Les expressions logiques ne seront utilisées qu'avec l'instruction **si**.

Syntaxe d'une expression logique : opérande1 oprel opérande2

Les expressions logiques concerneront donc toujours deux opérandes séparés par un opérateur relationnel (notation infixe). Un opérande est soit une constante, soit un identificateur.

L'opérateur relationnel oprel est un symbole parmi : = <> < <= > >=

- = représente le test d'égalité
- <> représente l'inégalité
- < infériorité stricte
- > supériorité stricte
- <= inférieur ou égal
- >= supérieur ou égal

Ces opérateurs ont le sens habituel pour les entiers.

Pour les chaînes, c'est l'ordre lexicographique habituel qui sera utilisé.

Exemples : "TATA" < "TU" (car A < U dans l'alphabet)
 "LONG" < "LONGUEUR" (la chaîne la plus longue est supérieure)

2 LES INSTRUCTIONS DE PROGRAMMATION

Les instructions de programmation permettent de faire effectuer des actions. Elles peuvent être utilisées comme commande immédiate ou comme ligne d'un programme (lorsque précédée par une <étiquette>).

1. Affectation : **var**

Syntaxe: **var** <identificateur>=<expression>

Sémantique : affecte la valeur de l'expression à la variable nommée par l'identificateur.

2. Entrée d'une valeur depuis le clavier: **entre**

Syntaxe: **entre** <identificateur>

Sémantique : attend que l'utilisateur entre une valeur sur l'entrée texte courante, l'évalue, puis l'affecte à la variable <identificateur>.

3. Sortie d'une valeur à l'écran : **affiche**

Syntaxe1 : **affiche** <expression>

Sémantique : évalue la valeur de l'expression et l'affiche sur la sortie texte courante.

Syntaxe2 : **affiche**

Sémantique : provoque un saut de ligne sur la sortie texte courante.

4. Branchement impératif : **vaen**

Syntaxe: **vaen** <étiquette>

Sémantique : continue l'exécution à partir du numéro spécifié par étiquette.

5. Branchement conditionnel : **si ... vaen**

Syntaxe: **si** <expression_logique> **vaen** <étiquette>

Sémantique : si la condition est vraie alors l'exécution continuera à partir du numéro de ligne spécifié par l'étiquette, sinon l'exécution continuera en séquence.

6. Appel d'une procédure : **procedure**

Syntaxe: **procedure** <étiquette>

Sémantique: L'exécution est transférée au numéro d'étiquette spécifié et reprendra en séquence lorsque la procédure sera terminée. Il n'y a pas de paramétrage de l'interface

=> la communication repose sur des « effets de bord » concernant les identificateurs.

=> les identificateurs déclarés sont donc tous à portée globale même s'ils ne semblent utilisés que dans une seule procédure (pas de principe de localité).

7. Fin d'une procédure : **retour**

Syntaxe: **retour**

Sémantique : cette instruction, rencontrée après un appel de procédure, provoque un retour à l'instruction qui suit son appel.

=> attention aux appels en cascade...

Remarque : une instruction **retour** ne peut être rencontrée que suite à une instruction **procedure** exécutée précédemment ; dans le cas contraire une erreur sera signalée.

8. Arrêt du programme : **stop**

Syntaxe: **stop**

Sémantique : Cette instruction arrête l'exécution du programme. Elle en marque donc la fin.

3 COMMANDES GENERALES

De façon à pouvoir manipuler plus facilement ses programmes, l'utilisateur disposera de commandes. Ces commandes ne sont pas des instructions et ne peuvent pas faire partie d'un programme.

3.1 *La commande de nettoyage de l'environnement: debut*

Syntaxe: **debut**

Sémantique : efface toutes les lignes de programme mémorisées ainsi que tous les identificateurs mémorisés.

3.2 *La commande d'effacement de ligne(s) : efface*

Syntaxe: **efface** <etiquette_debut>:<etiquette_fin>

Sémantique : efface toutes les lignes de programme dont le numéro d'étiquette est dans la plage comprise entre <etiquette_debut> et <etiquette_fin>.

3.3 *La commande d'affichage des lignes d'un programme : liste*

Syntaxe 1: **liste**

Sémantique : affiche toutes les lignes de programme mémorisées dans l'ordre croissant des numéros de ligne.

Syntaxe 2: **liste** <etiquette_debut>:<etiquette_fin>

Sémantique : affiche les lignes mémorisées dont les numéros d'étiquettes sont compris entre <etiquette_debut> et <etiquette_fin> (dans l'ordre croissant des numéros de ligne).

3.4 *La commande de liste des identificateurs définis : defs*

Syntaxe: **defs**

Sémantique : affiche la liste des identificateurs définis durant la session avec leur valeur (le programmeur voit donc le contexte du programme)

3.5 *La commande de lancement de l'exécution : lance*

Syntaxe 1: **lance**

Sémantique : démarre l'exécution d'un programme à partir de son plus petit numéro d'étiquette.

Syntaxe 2: **lance** <etiquette>

Sémantique : démarre l'exécution d'un programme à partir de l'étiquette spécifiée.

Remarque : une instruction **vaen** utilisée comme commande aura un effet similaire.

3.6 *La commande de sauvegarde dans un fichier : sauve*

Syntaxe: **sauve** cheminFichier

Sémantique : sauvegarde les lignes de programme dans le fichier texte indiqué en argument

3.7 *La commande de chargement depuis un fichier : charge*

Syntaxe: **charge** cheminFichier

Sémantique : charge dans le contexte les lignes de programme sauvegardées dans le fichier texte indiqué en argument

3.8 *La commande de fin de session : fin*

Syntaxe: **fin**

Sémantique : quitte l'interpréteur...

4 TRAVAIL A EFFECTUER

Réaliser un logiciel en langage java permettant à un programmeur d'utiliser un interpréteur de Langage LIR pour écrire et exécuter des programmes.

L'interpréteur affiche un caractère d'invite au programmeur afin qu'il puisse entrer une instruction ou une commande sur une ligne de commande.

Suivant la ligne de commande entrée, l'interprète :

- affiche un message d'erreur si la syntaxe de la commande ou de l'instruction ne permet pas de l'interpréter
- affiche un message d'erreur si la commande ou l'instruction conduit à une erreur d'exécution
- exécute la commande ou l'instruction puis affiche le résultat de la commande ou de l'instruction si elle produit un résultat à afficher
- exécute la commande ou l'instruction puis affiche « ok » en cas de commande ou d'instruction correcte ne générant pas de sortie particulière
- affiche « ok » en cas de mémorisation d'une ligne de programme correcte commençant par un numéro de séquence correct, indiquant ainsi que la ligne de programme a correctement été « insérée » dans la séquence.

Ainsi l'interpréteur donne toujours un « feedback » au programmeur lui permettant de savoir si la ligne de commande a été correctement interprétée ou s'il y a un problème. En cas d'erreur, un message expliquant le problème le plus clairement possible doit être affiché.

5 EXEMPLE D'UNE SESSION SOUS L'INTERPRETEUR LIR

(Lancement de LIR)

Interpréteur Langage IUT de Rodez, bienvenue !

Entrez vos commandes et instructions après l'invite ?

? charge bonjour.lir

ok

? liste

10 affiche "Entre ton nom : "

20 entre \$nom

30 affiche "Bienvenue "+\$nom

40 stop

? 40 var an=2021

ok

? 50 affiche "Quelle est ton année de naissance : "

nok : instruction inconnue : affiche

? 50 affiche "Quelle est ton année de naissance ? "

ok

? 60 entre

nok : paramètre obligatoire pour l'instruction entre

? 60 entre naissance

ok

? 200 stop

ok

? 70 affiche "Tu as autour de "

ok

? 65 si naissance > an vaen 50

ok

? 80 affiche an-naissance

ok

? 90 affiche "ans "

ok

? 100 affiche

ok

? 35 affiche

ok

```

? liste
10 affiche "Entre ton nom : "
20 entre $nom
30 affiche "Bienvenue "+$nom
35 affiche
40 var an=2021
50 affiche "Quelle est ton année de naissance ? "
60 entre naissance
65 si naissance > an vaen 50
70 affiche "Tu as autour de "
80 affiche an-naissance
90 affiche "ans "
100 affiche
200 stop
? lance
Entre ton nom : marc
Bienvenue marc
Quelle est ton année de naissance ? 2001
Tu as autour de 20 ans
? defs
an = 2021
naissance = 2001
$nom = "marc"
? sauve age.lir
Le programme age.lir a été sauvegardé.
? debut
ok
? liste
ok
? fin
Au revoir, à bientôt !

```