Université Paris Diderot
L2 Informatique et Math-Info

Analyse de Données Structurées
Année 2018-2019

Projet

Interpréteur pour un mini-robot

Dans ce projet vous allez concevoir et implémenter un interpréteur de programme pour un mini-robot se déplaçant sur une grille. L'interpréteur prendra en entrée deux fichiers, le premier contenant un programme dans le langage décrit par la grammaire définie ci-dessous, le second décrivant l'état initial de la grille sur laquelle le robot se déplace.

La grammaire La grammaire du langage est la suivante, les non-terminaux sont en gras et en italiques, les terminaux (jetons) en police `code` :

$$
\begin{aligned}
\textbf{\textit{programme}} &\to \textbf{\textit{instruction}}\ ;\ \textbf{\textit{programme}} \mid \varepsilon \\
\textbf{\textit{instruction}} &\to \texttt{Avancer(}\textbf{\textit{expression}}\texttt{)} \mid \texttt{Tourner(}\textbf{\textit{expression}}\texttt{)} \mid \texttt{Ecrire(}\textbf{\textit{expression}}\texttt{)} \\
\textbf{\textit{expression}} &\to \texttt{Lire} \mid \textbf{\textit{nombre}} \mid (\textbf{\textit{expression operation expression}}) \\
\textbf{\textit{operation}} &\to \texttt{+} \mid \texttt{-} \\
\textbf{\textit{nombre}} &\to \texttt{-?[1-9][0-9]*} \mid \texttt{0}
\end{aligned}
$$

La grille. Le robot se déplace sur une grille de X colonnes (numérotée de 0 à $X-1$ de l'ouest à l'est) et Y lignes (numérotées de 0 à $Y-1$ du sud au nord). Autrement dit : la case en bas à gauche est $(0,0)$ et la case en haut à droite est $(X-1, Y-1)$. Le robot commence sur la colonne x et la ligne y, et est orienté dans la direction d. L'état initial de la grille est décrite dans un fichier texte respectant le format suivant :

— Sur la première ligne se trouvent cinq entiers séparés par des espace : X Y x y d.
— Suivent Y lignes contenant chacune X entiers, les valeurs écrites sur chaque case de la grille.

On peut par exemple avoir la grille suivante :

```
5 10 0 0 0
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  2  0  0  0  4  0  0  0  0
0  0  0  0  0  0  0  0  0  0
5 -1  0  0  0  5  0  0  0  0
```

Comportement du robot. La grille est torique : si le robot essaie de se déplacer vers le sud en étant sur la ligne 0, il se retrouvera sur la ligne $Y-1$, *vice-versa* s'il essaie de se déplacer vers le nord en étant sur la ligne $Y-1$. De manière similaire, si le robot essaie de se déplacer vers l'ouest (resp. vers l'est) en étant sur la colonne 0 (resp. $X-1$), il se retrouvera sur la colonne $X-1$ (resp. 0), tout en restant sur la même ligne. À tout instant, le robot a une position (x, y) et une direction d (un entier défini modulo 4).

— Si $d \equiv 0 \pmod 4$, le robot est orienté vers l'est,
— Si $d \equiv 1 \pmod 4$, le robot est orienté vers le nord,
— Si $d \equiv 2 \pmod 4$, le robot est orienté vers l'ouest,
— Si $d \equiv 3 \pmod 4$, le robot est orienté vers le sud.

Un exemple de programme est donné ci-dessous.

```
Avancer(Lire); Tourner(Lire);
Avancer(7);    Tourner(-3);
Avancer(Lire); Tourner(1);
Avancer(Lire); Tourner((2-1));
Avancer(Lire); Ecrire((Lire+10));
```

Les instructions données au robot ont le comportement suivant :

- `Avancer`(***expression***) : Avancer le robot de ***expression*** cases dans la direction d.
- `Tourner`(***expression***) : Ajouter ***expression*** à la valeur de d.
- `Ecrire`(***expression***) : Écrire ***expression*** sur la case actuelle.

On évalue les expressions de manière naturelle, `Lire` étant égal à la valeur de la case actuelle. Le tableau ci-dessous détaille l'exécution du programme donné sur la grille d'exemple.

Instruction	x	y	d	Commentaire
	0	0	0	
`Avancer(Lire)`	5	0	0	`Lire` vaut 5
`Tourner(Lire)`	5	0	1	`Lire` vaut 5
`Avancer(7)`	5	2	1	
`Tourner(-3)`	5	2	2	
`Avancer(Lire)`	1	2	2	`Lire` vaut 4
`Tourner(1)`	1	2	3	
`Avancer(Lire)`	1	0	3	`Lire` vaut 2
`Tourner((2-1))`	1	0	0	
`Avancer(Lire)`	0	0	0	`Lire` vaut -1
`Ecrire((Lire+10))`	0	0	0	Écrit 15 sur la case (0,0).

1 Première étape

Ce projet doit être fait en JAVA. Un template de code vous est fourni. Vous pouvez le compiler avec `javac *.java` et le tester avec `java Main` ou `java Main programme.txt grille.txt`. Il est conseillé d'écrire le programme en plusieurs étapes de difficulté croissante. Dans un premier temps, vous devez écrire un analyseur lexical et syntaxique (une classe qui implémente l'interface `Parser`) afin de construire l'arbre de syntaxe abstraite d'un programme (la classe `Program`) décrit par la grammaire ci-dessus. Ensuite, écrivez un interpréteur pour l'arbre construit (une classe qui implémente l'interface `Interpreter`). Vous devrez fournir une série de tests permettant de vérifier que le comportement de votre programme est correct.

Gestion des erreurs. Votre programme doit traiter proprement toutes les erreurs qui peuvent se produire pendant l'interprétation, c'est à dire afficher un message explicite, détaillant l'erreur rencontrée, et puis terminer l'interpréteur.

2 Deuxième étape

On souhaite ajouter des conditions et des boucles à la grammaire du langage.

$$
\begin{aligned}
\textit{instruction} &\rightarrow \ldots \mid \texttt{Si}\ \textit{condition}\ \texttt{Alors}\ \textit{programme}\ \texttt{Fin} \\
&\quad \mid \texttt{TantQue}\ \textit{condition}\ \texttt{Faire}\ \textit{programme}\ \texttt{Fin} \\
\textit{condition} &\rightarrow \textit{expression}\ \textit{comparaison}\ \textit{expression} \\
&\quad \mid (\textit{condition}\ \textit{connecteur}\ \textit{condition}) \\
\textit{comparaison} &\rightarrow \texttt{<} \mid \texttt{>} \mid \texttt{=} \\
\textit{connecteur} &\rightarrow \texttt{Et} \mid \texttt{Ou}
\end{aligned}
$$

Implémenter ces fonctionnalités, en interprétant les conditionnelles et les boucles de manière naturelle. Attention, la grammaire ci-dessus n'est plus LL(1). Modifiez cette grammaire afin d'obtenir la rendre LL(1). Différentes solutions seront acceptées.

- Il est possible de légèrement modifier la grammaire, par exemple en changeant le type de parenthésage dans la règle ***condition*** $\rightarrow \ldots \mid \{$***condition connecteur condition***$\}$.
- Généraliser la grammaire, par exemple en regroupant les non-terminaux ***expression*** et ***condition***, ainsi que les opérations `+`, `-`, `<`, `>`, `=`, `Et`, `Ou`. Puis ajouter une phase d'analyse statique, permettant de "typer" les expressions obtenues et vérifier que l'arbre de syntaxe abstraite obtenu est valide.

Tant tout les cas, il vous est demandé d'expliciter clairement les choix effectués, et de fournir un ensemble de tests permettant de vérifier le bon fonctionnement de votre programme.

Afin de pouvoir vérifier le bon fonctionnement de votre programme, celui ci devra afficher les instructions effectuées.

Bonus (non noté). Écrire un programme pour le robot qui, en partant d'une grille dont tous les nombres inscrits sont strictement positif, se termine en ayant écrit sur une case de votre choix la somme de toutes les valeurs. Les valeurs des autres cases sont libres et peuvent être différentes des valeurs initiales. On pourra dans un premier temps traiter le cas d'une grille 1×2 puis $1 \times C$.

3 Troisième étape

Dans cette troisième étape, implémentez les fonctionnalités de votre choix ! Ci-dessous se trouve une liste (non-exhaustive) de possibilités.

- Ajouter sur la grille des blocs (représentés par un caractère `#`) que le robot n'est pas autorisé à traverser. Afin de pouvoir détecter un mouvement invalide, implementer un bloc ayant un comportement similaire au `try ... catch ...` en Java.

  instruction $\rightarrow \cdots \mid$ `Essayer` ***programme*** `Avec` ***programme*** `Fin`

- Implémenter une interface graphique permettant de visualiser les actions du robot.
- ...

4 Rendu

Il est conseillé de faire le projet en binôme (mais pas de trinômes !) Le projet est à rendre sur Moodle.

La date limite de rendu sera annocée une fois que le date de soutenance est connue.