Projet : Compilateur SLIP vers NBC
IHDC B332 - Théorie des langages : Syntaxe et sémantique

Nicolay Matthias
Evrard Cédric

UNamur

1 Introduction

Dans ce rapport, nous allons présenter les différentes aspects techniques du projet de syntaxe et sémantique ainsi que les choix techniques que nous avons effectués lors de la création du compilateur SLIP.

2 Démarche générale

Notre groupe était composé de deux personnes. La démarche que nous avons adopté a été de laisser Cédric commencer à travailler en chaque début de partie afin de créer l'architecture de l'application ainsi que la structure des différents packages. Une fois que cette structure était créée, une répartition était faite du travail.

Il a parfois été difficile de découper la réalisation du travail de manière équitable. En effet, une dépendance assez importante dans la grammaire de la partie *expression droite* à fait que celle-ci devait souvent être réaliser avant tout le reste et conditionnait énormément la structure du projet.

Avant la fin de la phase 2, un gros travail de refactoring a été effectué afin de rendre le code plus lisible et d'en faciliter la maintenance.

3 Structure de la table des symboles

La structure ustilisée se trouve dans la classe *Context.java*.

La "Table des symboles" en elle même est composée de quatre *Hashtables* qui sont accompagnées d'un *array* pour chanque symbole. Les quatre symboles différents sont :

- VariableBase
- Function
- Record
- Enumeration

Notre choix d'utiliser quatre *Hashtable* est simple, notre point de vue est qu'il y a quatre classe de symboles. Ansi donc pour avoir une simplicité de codage, de stockage et d'acces aux symboles, nous avons utilisé cette implémentation.

Notre "Table des symbole" est donc utilisée comme suit:

- Lors du passage de notre visiteur, chaque symbole est stoqué dans l'*array* assigné à son symbole et référencé dans sa *Hashtable*. Le tout dans le contexte dans lequel se trouve ce symbole.
- Lors du "test" du code, chaque *Function* est vérifiée par la validité de son type de retour et de la bonne présence des *VariableBase* dans leur contexte respectif pour qu'elles respectent leurs portées. Cette action est facilitée par la présence des *Hashtable*
- Chaque symbole est donc restitué en code NBC en conservant chaque fonctionalité du code *Slip* entré dans le compilateur.

Ceci est donc le résumé de l'implémentation de notra "Table des symboles".

4 Architecture du compilateur

Notre Compilateur est divisé en 6 *Packages*:

- Application : qui contient toutes les classes liées à la sauvegarde des symboles
- Exception : qui contient l'exception *PlayPlusException*
- Language : qui contient le visiteur du language et tous les *Helpers* qui aident le visiteur
- Main : qui contient la fonction main le parseur *ANTLR* et le *listener* d'erreurs
- Map : qui contient le *Listener* de la carte
- NBC : qui contient le *Printer* NBC, le *Visiteur* NBC, les *Helper* et *Writer* qui vont aider les deux premières classes

4.1 Application

Voici les diférentes classe présentes dans le *Package* Application.

4.1.1 Application

Cette classe permet d'initialiser une nouvelle instance d'*Application*, elle contient aussi les méthodes permetant d'entrer dans le bon context et d'en sortir. Elle contient aussi les méthodes d'ajout des différents symboles et celles qui permet d'y accèder.

4.1.2 Array

Cette classe permet d'initialiser un *Array* qui sera utilisé dans un context.

4.1.3 Context

Classe principale qui initialise un *Context* qui va être utilisé dans une *Application*. Ce *Context* contient nos 4 *Hashtable* dans lesquels nous sauvegardons nos symboles.

Ces 4 *Hashtables* sont les suivantes:

- variableSymbols : qui va contenir tous les symboles des variables.
- functionSymbols : qui va contenir tous les symboles des fonctions.
- recordSymbols : qui va contenir tous les symboles des records.
- enumSymbols : qui va contenir tous les symboles des énumérations.

Elles sont épaulées par 4 *Array*:

- variables : qui contient les différentes variables du programme.
- functions : qui contient les différentes fonctions du programme.
- records : qui contient les différentes structures du programme.

- enums : qui contient les différentes énumérations du programme.

Elle contient également la carte.

En plus de cela la classe contient les méthodes qui permettent d'ajouter/retourner les *Variable/Array/Function* et *Enum*.

4.1.4 Enumeration

Cette classe permet d'initialiser une *Enumeration*.

4.1.5 Function

Cette classe permet d'initialiser une *Function*. Elle *Extends* Context car elle doit pouvoir stocker les mêmes symboles qu'un contexte. En plus du contexte, elle doit stocker ses arguments et son type de retour. Toutes les méthodes liées à cette classe servent à ajouter/retourner les divers symboles contenus par cette fonction.

4.1.6 Record

Cette classe permet d'initialiser un *Record*.

4.1.7 VariableBase

Cette classe permet d'initialiser une *VariableBase*. Qui est le type de base d'une variable.

4.1.8 Variable

Cette classe *Extends VariableBase* et ajoute juste un attribut *isConstant* qui est là pour savoir si la variable est une constante où non.

4.2 Exception

Ce package ne contient que *PlayPlusException* qui *Extends RuntimeException* et qui est lancé quand il y a une exception non atendue dans le *Parser*.

4.3 Language

Ce package contient le *LanguageVisitor* qui va visiter l'arbre syntaxique et en extraire les symboles utiles. Le package contient aussi un dossier *Helper* qui contient toutes les méthodes qui vont être utilisées pour récupérer les symboles.

4.3.1 ActionExpression

Cette classe contient les méthodes qui servent à *parser* les *ActionExpression*.

4.3.2 ArrayHelper

Cette classe contient la méthode qui sert à convertir x noeuds terminaux *NUMBER* en un tableau d'entiers.

4.3.3 BooleanExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression Booléenne.

4.3.4 CharExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression de charactère.

4.3.5 ConstantExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression de constante.

4.3.6 DeclarationExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle Déclaration et Instruction.

4.3.7 EnumExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle énumération.

4.3.8 ForExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression *For*.

4.3.9 FunctionExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression de fonction, argument, appel de fonction où déclaration de fonction.

4.3.10 GenericExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression gauche, de comparaison où de parenthèses.

4.3.11 IfExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle instruction *If*.

4.3.12 IntegerExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression d'*Integer*.

4.3.13 RepeatExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression *Repeat*.

4.3.14 StructureExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle structure.

4.3.15 Tuple

Cette classe permet de créer un *Tuple* de deux items de type différents.

4.3.16 VariableExpression

Cette classe contient les méthodes qui servent à *parser* les instructions, les définitions, les initialisations de variables et de tableaux.

4.3.17 VariableHelper

Cette classe contient les méthodes qui servent à ajouter une variable, un tableau et une structure. Elles sont utilisées dans *VariableExpression* et *ConstantExpression*.

4.3.18 WhileExpression

Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression *While*.

4.4 Main

Ce package contient la fonction *Main* le *paser ANTLR* et le *listener* d'erreurs.

4.5 Map

Ce package ne contient qu'une classe qui est le *MapListener* qui va *parser* la carte , sauvegarder ses symboles dans un tableau et vérifier qu'elle est correcte.

4.6 Nbc

Ce package contient deux classes principales *NBCVisitor* et *NBCPrinter* qui vont respectivement visiter notre programme et le convertir en instuctions *NBC*. Les classes aidant sont réparties en deux packages.

4.6.1 Helper

Premier package qui contient toutes les classes d'aide.

ActionExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle action en code *NBC*.

ActionInterface Classe qui permet d'exécuter le *Writer*.

ArrayExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle *Array* en code *NBC*.

AssignationExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle assignation en code *NBC*.

ComparisonExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle comparaison en code *NBC*.

DeclarationExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle déclaration en code *NBC*.

ForExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle *For* en code *NBC*.

FunctionExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle fonction en code *NBC*.

IfExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle *If* en code *NBC*.

IntegerExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression entière en code *NBC*.

IntegerHelper Cette classe sert à renvoyer le bon symbole lors d'une opération entière.

LeftExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression gauche en code *NBC*.

MapExpression Cette classe contient les méthodes qui servent à importer la carte.

RepeatExpression Cette classe contient les méthodes qui servent à *parser* n'importe quel *Repeat* en code *NBC*.

RightExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression droite en code *NBC*.

VariableExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle variable en code *NBC*.

VariableHelper Cette classe sert à renvoyer le bon symbole dépendant le type de variable.

WhileExpression Cette classe contient les méthodes qui servent à *parser* n'importe quelle expression *While* en code *NBC*.

4.6.2 Writer

Ce package contient toutes les classes servant à écrire le code *NBC*.

ActionWriter Cette classe permet d'écrire n'importe qu'elle type d'action.

FunctionWriter Cette classe permet d'écrire n'importe qu'elle type de fonction.

IfWriter Cette classe permet d'écrire n'importe qu'elle type de *If*.

LogicWriter Cette classe permet d'écrire n'importe qu'elle type de comparaison logique.

LoopWriter Cette classe permet d'écrire n'importe qu'elle type de boucle.

NBCCodeTypes Cette classe renvoie la bonne écriture du type de varaible en code *NBC*.

NBCIntCodeTypes Cette classe renvoie la bonne écriture du type d'oppération entière en code *NBC*.

NBCOpCodeTypes Cette classe renvoie la bonne écriture du type d'opérateur logique en code *NBC*.

NBCWriter Cette classe écrit le code *NBC* de base inérent au déplacement du robot.

OperationWriter Cette classe permet d'écrire n'importe qu'elle type d'opération en code *NBC*.

PreprocessorWriter Cette classe permet d'écrire le préprocess en code *NBC*.

VarialbeWriter Cette classe permet d'écrire n'importe qu'elle type de varaible en code *NBC*.

5 Conclusion

References