REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediène

---

Projet de TP :

Réalisation d'un mini compilateur pour le langage 'TinyLanguage_SII'

Avec l'outil ANTLR

I. Motivation

Un compilateur est un programme qui traduit un programme écrit dans un langage source vers un langage cible en indiquant les erreurs éventuelles que pourrait contenir le programme source. Un informaticien à peu de chances d'être impliqué dans la réalisation d'un compilateur pour un langage de programmation majeur.

✓ **Alors pourquoi ce TP ?**

La compilation n'est pas limitée à la traduction d'un programme informatique écrit dans un langage de haut niveau en un programme directement exécutable par une machine, cela peut aussi être :

- La traduction d'un langage de programmation de haut niveau vers un autre langage de programmation de haut niveau.
- La traduction d'un langage de programmation de bas niveau vers un autre langage de programmation de haut niveau. Par exemple pour retrouver le code C à partir d'un code compilé (piratage, récupération de vieux logiciels, etc).
- La traduction d'un langage quelconque vers un autre langage quelconque (i.e. pas forcément de programmation) : word vers html, pdf vers ps, etc.

Ce genre de travail peut très bien être confié à un ingénieur maître de nos jours.

La compilation se décompose en deux phases :

- Une phase d'analyse, qui va reconnaître les variables, les instructions, les opérateurs et élaborer la structure syntaxique du programme ainsi que certaines propriétés sémantiques.
- Une phase de synthèse et de production qui devra produire le code cible.

Pour réaliser un compilateur, plusieurs environnements sont disponibles :

- Lex/Yacc,
- Flex/Bison,
- Java Compiler Compiler,
- Free Compiler Construction Tools,
- ANTLR, ANother Tool for Language Recognition,
- The Compiler Generator Coco/R, etc.

Dans le cadre de ce projet, nous allons utiliser ANTLR (Another Tool for Language Recognigtion), prononcé antler, est un générateur d'analyseurs qui utilise LL(*) pour l'analyse syntaxique. C'est un outil qui propose un framework pour construire des compilateurs à partir de descriptions grammaticales qui peuvent éventuellement contenir des instructions écrites en Java, C++, C#, etc. Un générateur d'analyseurs est un outil qui lit une grammaire en entrée et la convertit en un programme qui peut reconnaître un texte et le traiter suivant les règles de cette grammaire. ANTLR a été développé par Terrence Parr dans le langage Java, mais il peut générer des analyseurs dans un code écrit dans l'un de ses nombreux langages de programmation cibles. Sa dernière version est ANTLR 4.9 qui génère un code Java, C#, Python2, Python3, JavaScript, Go, C++ et Swift.

En plus du générateur d'analyseur, ANTLR fournit d'autres fonctionnalités y afférentes telles que la construction d'arbres, l'insertion des actions dans les règles de grammaire, la gestion d'erreurs et le débogage. Il existe un environnement d'utilisation graphique appelé AntlrWorks. ANTLR et AntlrWorks sont libres et open source. Plusieurs outils de développement peuvent être utilisés tel que : plug-ins for Intellij, NetBeans et Eclipse.

Un manuel détaillé de cet outil a été mis à la disposition des étudiants.

TinyLanguage_SII

La définition informelle du langage TinyLanguage_SII est comme suit :
// Les commentaires sont sur une seule ligne comme celui-ci

/* Les commentaires sont sur
plusieurs lignes comme celui-la */

// Les mots clé sont en **gras** minuscules
// Le nom du programme commence par une lettre majuscule
// Les identificateurs commencent par une lettre éventuellement suivie de lettres ou de chiffres
// On distingue les majuscules et les minuscules : a et A ne sont pas le même identificateur
// La structure générale d'un programme TinyLanguage_SII

```
compil Nom_du_programme ()
{
 // Partie déclaration des variables
 start
 // Partie description du programme
}
```

// Les instructions sont terminées par des points virgules ;

// Déclaration des variables

```
intCompil term1, term2 ;
floatCompil Div ;
StringCompil chaine1 ;
```

// Les opérateurs des expressions numériques sont : +, -, *, /
// Leurs opérandes sont des nombres entiers ou réels
// Le type de l'expression est entier si tous ses opérandes sont entiers, réel sinon
// On peut comparer des nombres entre eux par : >, <, ==, !=
// La priorité des opérateurs est comme suit , par ordre croissant : /, *, +, -, >, <, == , != sauf le cas des parenthèses.
// L'affectation (=) de variables avec des valeurs entières ou réelles

```
Div = 0 ;
```

// Les valeurs affectées ne sont pas obligatoirement des constantes, mais sont le résultat de l'évaluation d'expressions syntaxiquement et sémantiquement correctes :

```
term2 = term1+1 ;
```

// L'instruction conditionnelle **if-then-else (la partie sinon est optionnelle)**

```
if (term1 > 0) then {
                term2 = term2 + 1 ;
                term1 = term1+1 ;
        }else{
              term2 = term2 + 2 ;
              term3 = term3 / term2 ;
              }
```

// La boucle **do - while (la condition)**

```
        do {
              term2 = term3 + 1 ;
              term1 = term1+1 ;
              term3 = term3 / term2 ;
             }while(term1 != 10)
```

// L'instruction de l'ecture

scanCompil (term1, term2) ;

// term1 et term2 auront le type de ce qu'on lit sur l'entrée.

// L'instruction d'écriture

printCompil (Le résultat est) ;
printCompil (Div) ;

II. Travail à réaliser

Vous aurez à réaliser, à l'aide de "ANTLR", un compilateur du langage **TinyLanguage_SII** dont la description vous a été fournie dans la section précédente. Le code produit sera le code objet en **assembleur**.

Pour cela, vous devez :

- ✓ Définir la grammaire adéquate.
- ✓ Construire un analyseur lexical en utilisant "ANTLR".
- ✓ Construire un analyseur syntaxique en utilisant "ANTLR".
- ✓ Les deux analyseurs seront complétés par l'analyse sémantique associée au programme source analysé.
- ✓ Il est demandé d'afficher les messages d'erreurs adéquats pour chaque étape.
- ✓ L'analyse sémantique intervient ensuite afin d'effectuer les **vérifications nécessaires** à la sémantique du langage et **mis à jour** la table des symboles.
- ✓ Le code intermédiaire généré doit être sous forme de quadruplets.
- ✓ Enfin, Génération du code objet en langage Assembleur.

Gestion de la table de symboles :

La table de symboles doit être créée lors de la phase de l'analyse lexicale. Elle regroupe l'ensemble des variables avec différents types définies dans le code source avec toutes les informations nécessaires pour le processus de compilation. Cette table sera mise à jour au fur et à mesure de l'avancement de la compilation.

Il est demandé de prévoir des fonctions permettant la recherche et l'insertion des éléments dans la table des symboles. La table doit avoir au minimum les champs suivants :

- Nom : l'identificateur qui indique le nom de la variable.
- Type : le type de la variable.
- Valeur : contient la valeur de la variable.
- Déclarée : pour vérifier la déclaration des variables dans la partie déclaration.

Vous pouvez éventuellement rajouter des champs à la table des symboles que vous jugez nécessaires.

III. Evaluation du projet

- Le travail demandé peut-être individuel ou en binôme à condition de travailler en équipe et chaque étudiant doit rajouter une valeur.
- Le projet sera évalué progressivement pendant chaque séance.
- Le projet et le compte rendu détaillant toutes vos étapes, seront remis juste avant les examens (la date exacte sera communiquée par la suite).
- Une appréciation sur le travail fait sera donnée par l'assistante à la fin de chaque séance de TP, qui comptera dans la note finale du TP.
- Il ne sera pas toléré le passage de programmes entre étudiants. Tous projets ressemblants seront sanctionnés incluant le développeur réel du projet.