UNIVERSITÉ DE NANTES

Université de Nantes

Compilation

Luthor Cowp

Compilateur FreeBASIC vers C

Étudiants :
Maxime Pauvert
Nicolas Brondin

Encadrant :
Benoît Guédas

Avril 2015

Table des matières

Chapitre 1

Introduction

Dans le cadre du module de Compilation, nous avons à réaliser un projet individuel ou en binôme.

1.1 Objectifs

Le but de ce projet est de créer un compilateur prenant un fichier écrit en FreeBASIC et générant un fichier C.

Il est impossible dans le temps imparti de créer un compilateur exhaustif, le but de ce module était d'arriver à implémenter le plus de règles possible pour le compilateur.

1.2 Contraintes

- Langage de programmation : OCaml
- Outils : OCamlLex et OcamlYacc
- Langage source : FreeBASIC
- Langage destination : C
- Le plus de règles possibles
- Compilable sous linux

1.3 Composition d'un compilateur

Un compilateur est composé de deux principales parties : Un analyseur lexical et un analyseur syntaxique. L'analyseur lexical permet de lire le flux d'entrée et de spécifier le lexique grâce à des expressions régulières.

Si un "mot" ne correspont à aucune des expressions régulières, une erreur lexicale est renvoyée. Une fois qu'un mot du lexique est trouvé, le token correspondant est passé à l'analyseur syntaxique. L'analyseur syntaxique lui va analyser l'ordre dans lequel les tokens vont lui être passées. Tant que l'ordre correpond aux rêgles syntaxiques du langage source, la traduction vers le langage cible est effectué, sinon une erreur de syntaxe est renvoyée.

1.4 Règles implémentées et ordre d'implémentation

Les différentes règles ont été implémentées une par une, voici l'ordre dans lequel est l'ont été :

- EOF

– EOL
– PRINT
– DIM AS STRING INTEGER
– ERROR
– IF THEN ELSE
– FOR WHILE DO LOOP NEXT
– OPERATOR
– OPERATION
– CONST

Chapitre 2

Scanner

2.1 Expressions Régulières

La plupart des expressions régulières pour scanner le langage FreeBASIC sont simplement les mots-clés complets du langage, mais certaines expressions régulières un peu plus complexes, le détail de leur fonctionnement est décrit ci-après :

2.1.1 identifier

```
['a'-'z' 'A'-'Z' '_'] ['a'-'z' 'A'-'Z' '0'-'9' '_']*
```

2.1.2 string

```
'"' [^'"']* '"'
```

2.1.3 integer

```
['0'-'9']+
```

2.1.4 number

```
integer ('.' integer)?
```

2.1.5 operation

```
['0'-'9' '+' '-' '*' '/' '(' ')']+
```

Chapitre 3

Parser

Chapitre 4

Conclusion

4.1 Travail restant

Dû à de nombreux problèmes, notre avancée sur le projet a été très lente, pour se débloquer vers la fin, il y a donc certaines fonctionnalités essentielles qui ne sont pas présentes :

- Gestion des commentaires
- Gestion des fonctions
- Gestion des erreurs dû aux types des variables

4.2 Les côté positifs

Le vrai côté positif de ce projet, et c'est normal, est d'avoir pu apprendre à se servir des outils Lex et Yacc et surtout d'avoir une première expérience réelle sur la compilation.

4.3 Les côté négatifs

Notre ressenti global sur le projet est que le séances de TP auraient dû être regroupées car 1h30 c'est trop peu pour pouvoir se remettre dans le projet et pouvoir avancer correctement.

Le gros point noir de ce projet reste le langage utilisé (OCaml) dans lequel nous n'avions aucune expérience alors qu'il aurait été beaucoup moins compliqué et chronophage pour nous d'utiliser les implémentations Java de Lex et Yacc.