UNIVERSITÉ DE NANTES

Université de Nantes

Compilation

Luthor Cowp

Compilateur FreeBASIC vers C

https ://github.com/masters-info-nantes/luthor-cowp.git

Étudiants :
Maxime Pauvert
Nicolas Brondin

Encadrant :
Benoît Guédas

Avril 2015

Table des matières

Chapitre 1

Introduction

Dans le cadre du module de Compilation, nous avions 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 correspond 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 passés. 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 Ordre d'implémentation des fonctionnalité

De nombreux soucis nous ont obligé à recommencer entièrement le projet à quelques jours du rendu, voilà pourquoi le planning suivant ne contient que l'ordre d'implémentation et pas les numéros des semaines.

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

- Reconnaissance de la fin du fichier
- Reconnaissance de la fin des lignes
- Gestion des PRINT
- Gestion des types et des déclarations de variables
- Gestion des erreurs syntaxiques
- Gestion des conditions
- Gestion des boucles
- Gestion des opérateurs de comparaison
- Implémentation de la Hastable des variables
- Gestion des opérations
- Gestion des constantes

Chapitre 2

Scanner

2.1 Expressions Régulières

En plus d'avoir à reconnaitres les mots-clés du FreeBASIC, il nous a fallut créer des expressions régulières un peu plus complexes pour certaines données, le détail de leur composition est décrit ci-après :

2.1.1 Les identifieurs

L'expression régulière suivante :

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

permet de trouver les identifieurs. Un identifieur est un nom de variable ou de fonction, il peut être composé d'autant de chiffres, de lettres et d'underscore que l'on veut, sauf pour le premier caractère qui ne peut pas être un chiffre.

2.1.2 Les chaines de caractères

L'expression régulière suivante :

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

permet de trouver une chaine de caractère qui est une suite d'autant de chiffres, lettres et caractères spéciaux possible mais qui doit obéir à deux règles :

- Doit commencer et se terminer par des doubles quotes
- Ne doit pas contenir de double quote (sauf échappement)

2.1.3 Les entiers

L'expression régulière suivante :

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

est la plus simple et permet de sélectionner les entiers, contennants au minimum un chiffre.

2.1.4 Les nombres

L'expression régulière suivante :

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

est dérivée de la RegExp précédente et permet de sélectionner les nombres à virgules.

2.1.5 Les opérations

L'expression régulière suivante :

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

permet de sélectionner toutes les opérations, une opérations se définissant comme une expressions contenant seulement des chiffres et des opérateurs.

Chapitre 3

Parser

3.1 Les règles

3.1.1 main

La règle main est la première règle à être exécutée, c'est le point de départ. A la fin de cette dernière, donc à la fin du fichier (EOF), on pourra alors écrire la fin du fichier de sortie qui se compose simplement d'un return 0 et d'une accolade fermante.

3.1.2 start

La règle start est exécutée immédiatement après la règle main et c'est elle qui va pouvoir écrire le début du fichier de sortie, c'est à dire les inclusions de librairies et la fonction main en C.

3.1.3 expressions

La règle expressions est celle qui sera appellée la plus souvent et qui va servir à appeler toutes les règles selon le mot qui aura été scanné.

3.1.4 loop

La règle loop est, comme son nom l'indique, la règle qui gère toutes les boucles, les for et les while. Elle gère le début mais aussi la fin des boucles avec les mots clés NEXT et LOOP. Comme les FOR en FreeBASIC ne contiennent aucun prédicat compliqué, ils sont géré de manières plus simplifiés que les WHILE qui eux peuvent avoir des conditions plus complexes.

3.1.5 condition

Derrière la règle condition se cache la gestion des IF, ELSE IF et ELSE. Pour chaque condition (sauf ELSE), on vient vérifier la présence et la conformité du prédicat.

3.1.6 predicate

La règle predicate vérifie si le prédicat effectue bien une comparaison entre un identifieur et un nombre, un string ou un autre identifieur.

3.1.7 print_expr

Pour écrire dans la console un texte ou le contenu d'une variable, on utilise la règle print_expr. Les variables étant stockées dans une Hashmap, la règle est capable de savoir le type de la variable et ainsi de générer correctement le printf en découlant.

3.1.8 definitions

Lors de la déclaration d'une variable, en plus de traduire le langage, on stocke la variable et son type dans une Hashmap, ce qui nous permet soit de retrouver le type de cette variable plus tard, soit tout simplement de savoir si la variable a bien déjà été déclarée avant de l'utiliser.

3.1.9 initialisations

Le but de la règle initialisation est de vérifier que l'on attribue bien soit une valeur brute à une variable, soit le résultat d'une opération.

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
- Gestion des erreurs sur les calculs

4.2 Les côtés 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 côtés 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.