Compilation : Lex et Yacc

Léo Cassiau

Mars 2016

1 Introduction

Depuis maintenant plusieurs années en informatique, la très grande majorité des développeurs utilisent des compilateurs pour créer des programmes. Mais comment sont construits ces compilateurs qui transforment notre code écrit dans un langage de haut niveau, en un code écrit dans un langage compréhensible uniquement (ou presque) des ordinateurs ?

La complexité des programmes grandissants, il est devenu trop complexe de créer un compilateur de toutes pièces, et c'est ainsi que sont apparus les compilateurs de compilateur. En général, ces compilateurs transforment donc un code représentant la grammaire du langage cible, en code compilant un programme écrit dans le langage cible. En définitive, le C, le Java ou encore le Python sont des langages définis par leurs compilateurs respectifs, et ces compilateurs sont créés de compilateur de compilateur.

Nous allons donc nous intéresser à ces compilateurs de compilateur, et plus particulièrement au couple le plus populaire dans le domaine, Lex et Yacc. En effet, ces deux programmes fonctionnent souvent de paire afin de créer un compilateur de compilateur. La première partie de ce document s'attarde donc sur Lex tandis que la deuxième partie s'intéresse à Yacc. Enfin, une conclusion clore se rapport.

2 Lex

Lex est un outil permettant de générer un analyseur lexical en C. Il génère donc un programme analysant une chaîne de caractères (une grammaire) afin de le décomposer en entité lexicale : des tokens. Lex est donc un outil permettant de créer l'équivalent de la classe ScannerToken et de la fonction analyse de la classe GrammaireZero qui sont dans notre projet.

Lex fonctionne de la manière suivante : il prend en entrée un fichier d'extension .l, qui définit les règles formant un token, et il génère un automate déterministe à états finis sous la forme d'un code C d'extension .lex.yy.c, automate qui reconnaît les tokens précédemment définis dans une chaîne de caractères donnés.

Le fichier .l de Lex utilise un langage spécifique à Lex qui permet de facilement définir la forme des tokens. On peut par exemple reconnaître des expressions régulières ou ignorer les blancs et les tabulations. C'est là l'intérêt principal de Lex : il permet de créer un analyseur lexical aisément compréhensible et facile à modifier, permettant ainsi de reconnaître des tokens de formes très variés, dans le but de rendre le compilateur plus facile à utiliser. Ainsi, en utilisant Lex durant notre projet, on aurait pu utiliser d'autres types de délimiteurs entre les différents tokens que l'espace, et ainsi rendre la grammaire plus facile à écrire.

En définitive, un générateur d'analyseur lexical comme Lex est un outil indispensable pour créer un compilateur de compilateur puissant. En effet, il est possible avec un simple parcours de fichier de donner un résultat similaire, mais cela s'avère beaucoup plus laborieux et difficilement maintenable. Mais l'analyse lexicale n'est que la première étape pour compiler un compilateur, il nous faut donc un autre outil pour accomplir la seconde étape, l'analyse syntaxique, et c'est là que Yacc intervient.

3 Yacc

Yacc est un générateur d'analyseur syntaxique en C. Il génère donc un programme analysant une chaîne de caractères (une grammaire) afin de d'exécuter les actions générant la grammaire du futur compilateur (et non du compilateur de compilateur). Yacc est donc un outil permettant de créer l'équivalent de la classe GrammaireZero, principalement des fonctions genForet, et analyse de cette classe, et aussi de la fonction GplActions de la classe GrammaireGpl.

Yacc fonctionne de la manière suivante : il prend en entrée un fichier d'extension .y qui définit la grammaire du compilateur (et non du compilateur de compilateur) (GenForet dans notre programme) et ses actions sémantiques (Gplactions dans notre programme). Puis, Yacc génère un programme C d'extension y.tab.c analysant un code à compiler.

Le fichier .y de Yacc comprend donc une grammaire, celle du futur compilateur, et les actions associées à la grammaire. Cette grammaire doit donc respecter la syntaxe permettant d'écrire des grammaires en Yacc, et les actions sont définis à la fin de chaque règle, rendant les actions plus simple à lire que si elles étaient situées dans un autre fichier par exemple. Ci-dessous un exemple de grammaire et de ses actions illustre la syntaxe de Yacc, les actions étant entre accolades à droite tandis que la grammaire est définie à gauche (Source : http ://pageperso.lif.univ-mrs.fr/ laurent.braud/compilation/lex_yacc.pdf).

```
calcul: expression                  {printf("%dn",$1);}
    ;
expression: expression '+' terme    {$$ = $1 + $3;}
    | expression '-' terme          {$$ = $1 - $3;}
    | terme                         {$$ = $1;}
    ;
terme: terme '*' facteur            {$$ = $1 * $3;}
    | terme '/' facteur             {$$ = $1 / $3;}
    | facteur                       {$$ = $1;}
```

```
    ;
facteur: '(' expression ')'          {$$ = $2;}
    | '-' facteur                    {$$ = -$2;}
    | NOMBRE                         {$$ = $1;}
    ;
```

Finalement, un générateur d'analyseur syntaxique comme Yacc permet de définir la grammaire du futur compilateur, et les actions du compilateur, le tout en privilégiant la simplicité et la lisibilité. En effet, le fait d'écrire les actions à coté des règles rend certainement le code plus logique et facile à déboguer, contrairement à notre fonction GplAction qui est plus obscure.

4 Conclusion

L'association de Lex et Yacc permet de réaliser rapidement des compilateurs efficaces, et l'apparition de compilateur de compilateur comme le couple Lex et Yacc est sûrement la raison de la multitude de nouveaux langages de haut niveau auxquels on a accès aujourd'hui. En effet, lors de notre projet nous avons pu voir à quel point il était difficile de définir un compilateur tout en maintenant un code et une grammaire lisible et cohérente. Or, Lex et Yacc permettent de rendre la grammaire, les actions et les tokens plus homogène, lisible et facile à modifier. Finalement, ce projet nous aura permis de mieux cerner le rôle et les atouts de Lex et Yacc, tout en nous formant sur le processus de création d'un compilateur.