Compilateur Lua - Projet IFT3150

Philippe Caron

29 août 2017

1 Introduction

L'objectif de ce travail est de se familiariser avec la compilation de langages impératifs en écrivant un compilateur pour le langage Lua. Lua signifie Lune en Portugais, c'est pourquoi le nom «Luna» a été choisi pour le compilateur. C'est un langage de script normalement interprété par un programme écrit en C. C'est un langage relativement simple dont la principale fonction est de fournir un outil de programmation multi-plateforme pour les logiciels complexe, c'est-à-dire ceux qui permettent que de nouvelles fonctionnalités soient créées par l'utilisateur, tels que les logiciels de modélisation 3D ou d'édition graphique, ainsi que pour les système embarqués.

2 Objectifs

2.1 Objectif principal

Le projet sera considéré comme un succès si le compilateur répond aux exigences suivantes :
- Le compilateur est écrit en lua
- Le compilateur produit du code assembleur compilable par gcc (syntaxe GAS) pour l'architecture Intel X86-64 sur un ordinateur linux ou mac, sans erreur.
- Le compilateur fonctionner sur un sous-ensemble de lua défini ci-bas
- Le compilateur devra avoir un système de gestion automatique de la mémoire (*garbage collector*)

2.1.1 Sous-ensemble de compilation

Le compilateur devra prendre en charge tous les éléments suivants :
- les boucles de lua, soient «while», «for», «for ... in», et «repeat ... until»
- la définition de fonctions
- les variables locales
- les tables
- les nombres sous forme d'entiers
- les chaînes de caractères
- tous les opérateurs de base (+, -, *, /, ; ==, =, =, <=, >=, <, >, and, or, not)
- les conditions («if»)
- les fonctions suivantes :
  - io.write()
  - io.read()
  - print()
  - pairs()
  - ipairs()

2.2 Objectifs secondaires

Sans qu'ils soient essentiels au succès du projet, celui-ci sera tout de même développé en gardant en tête certains objectifs secondaires. Premièrement, le code assembleur produit devra si possible respecter les

conventions d'appel de C, de sorte que les fonctions écrites en lua compilées avec le compilateur pourront être liées à du code C compilé par gcc et vice-versa. De plus, le compilateur sera écrit avec l'espoir de pouvoir être compilé par lui-même, ce qui en ferait un compilateur autogène. Afin de maximiser les chances de réaliser cet objectif un minimum de fonctions prédéfinies devra être utilisé lors de l'écriture du compilateur.

2.3 Objectifs facultatifs

Ces objectifs ne sont définis que dans le cas où le projet serait achevé avant la date d'échéance. À ce moment, les principaux efforts supplémentaires seront mis dans :

— L'écriture d'un assembleur (dont le vocabulaire se limitera à celui utilisé par le compilateur)
— Le support des nombres à virgule

3 Structure

Le compilateur sera diviser en plusieurs programmes qui seront exécutés tour à tour sur des fichiers intermédiaires. Ceci vise à rendre le processus plus clair ainsi qu'à augmenter la modularité du produit fini.

3.1 Préprocesseur

Le préprocesseur se chargera de retirer les commentaires du code, bien indenter le code, retirer les macros et/ou le sucre syntaxique. Il prendra en entré un fichier .lua et produira un fichier .pp.lua.

3.2 Compilateur intermédiaire

Le compilateur intermédiaire convertira le code lua en code intermédiaire (qui sera défini ultérieurement). La représentation intermédiaire a pour but de simplifier la lecture pour le compilateur final, rapprocher la représentation du code avec celle de la machine (en mettant en évidence une pile par exemple), ainsi que de permettre une modularité accru au niveau du compilateur. Il prends les fichiers .pp.lua produits par le préprocesseur et envoie des fichiers .lir au compilateur principal.

3.3 Compilateur final

Le compilateur final produira du code assembleur à partir de la représentation intermédiaire. Il prend donc les fichiers .lir en entré et sort des fichier .s.

3.4 Assembleur

Jusqu'à nouvel ordre, l'assembleur sera gcc. Il doit pouvoir prendre en argument les fichiers .s et .o et produire un fichier .exe fonctionnel.