Mình sẽ chia sẻ với các bạn tổng quan về compiler, và thử implement một phần trong compilation.

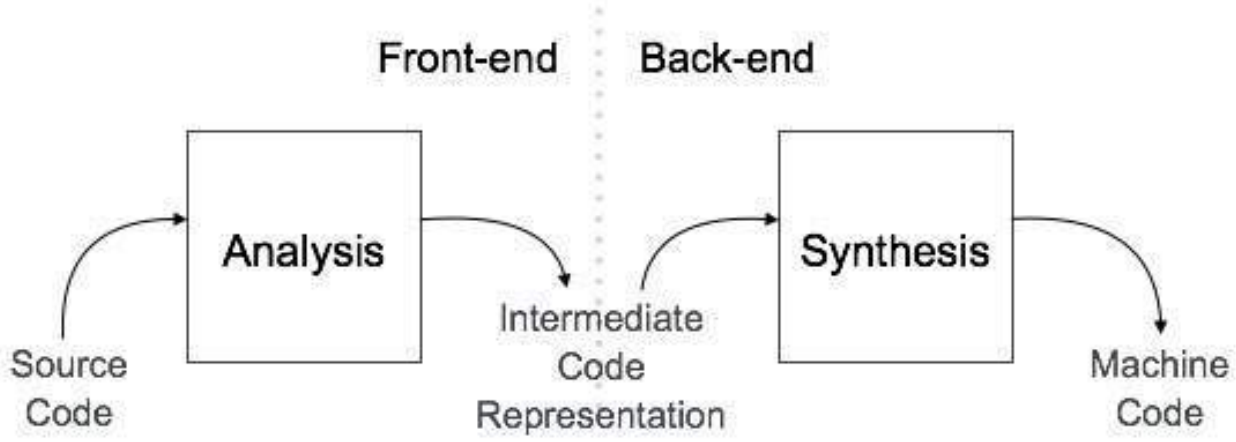
Tổng quan về compiler

Như các bạn đã biết, mã máy(machine code) là một loại ngôn ngữ lập trình mà chỉ bao gồm hai con số 0 và 1. Với khả năng readable gần như bằng 0, ngôn ngữ máy không thể dùng phổ biến cho việc lập trình. Điều này dẫn tới việc ra đời của các ngôn ngữ cấp cao. Đặc điểm của hướng tiến hóa là càng ngày càng gần với ngôn ngữ tự nhiên. Tuy nhiên, machine code là thứ duy nhất mà CPU có thể hiểu được. Vì vậy, với mỗi loại ngôn ngữ sinh ra, đều có đi kèm song song một công cụ để biên dịch từ mã nguồn cấp cao sang mã nguồn cấp thấp có thể execute được. Đó là compiler.

Vậy định nghĩa compiler là gì? Là một chương trình giúp biên dịch từ một ngôn ngữ bậc cao xuống một ngôn ngữ cấp thấp(thường là machine code).
Ngược lại với compiler là decompiler, giúp dịch ngược từ một ngôn ngữ bậc thấp lên một ngôn ngữ bậc cao. Nghe giống reverse engineering? Chính xác thì decompilation chỉ là một kĩ thuật trong reverse engineering.
Ngoài ra còn có transpiler – dịch từ một ngôn ngữ bậc cao sang một ngôn ngữ bậc cao khác. Ví dụ như babel trong JS, là một trình giúp dịch từ ES6 xuống ES5.

Cách hoạt động của một compiler

Một compiler khi hoạt động sẽ có 2 phases: frontend và backend. Theo một số tài liệu thì họ chia ra thêm 1 phase là middle end nằm chính giữa nhưng theo mình thì 2 phases là đủ.



Frontend compiler: Ở phase này, mã nguồn cấp cao sẽ được transform thành IR(Intermediate Representation). Các bạn tạm hiểu đây là một data structure nằm trung gian giữa front end và backend compiler, mà được thiết kế để dễ dàng hơn trong việc optimize và translate. Một IR phải có độ chính xác tuyệt đối(có thể represent mã nguồn mà không mất thông tin). Các công việc sẽ làm ở front end bao gồm:

- Phân tích mã nguồn, kiểm tra cú pháp và ngữ nghĩa
- Type checking
- Sinh ra error và warning nếu có.

Backend compiler sẽ nhận vào output của front end. Công việc chính của backend sẽ là code optimization và code generation. Vậy output của backend compiler là gì?

Backend compiler của các ngôn ngữ sẽ produce ra các loại output khác nhau. Có 3 loại output chính:

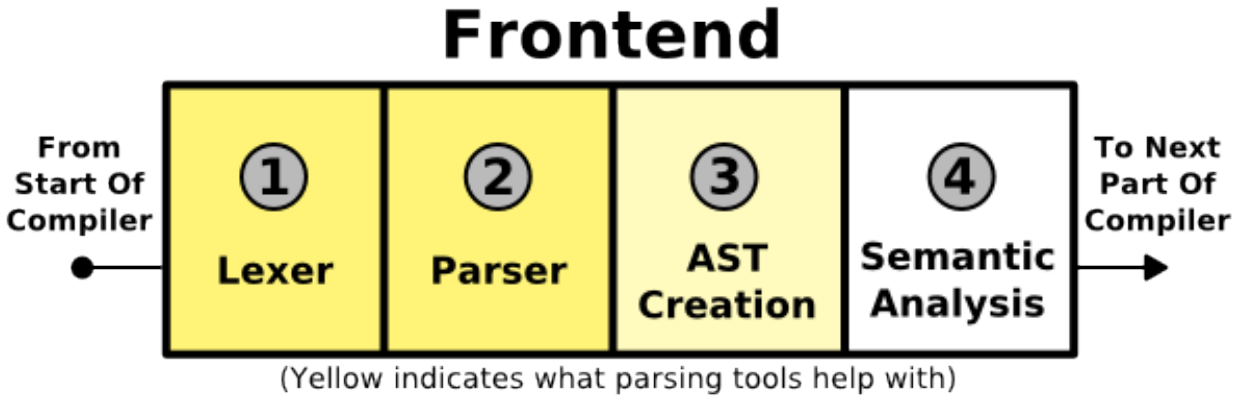
- Assembly code.(gcc của C)
- Bytecodes. (javac của Java, Smalltalk)
- Machine code. (Go compiler, MS compiler của C)

Sở dĩ có sự khác nhau này là do cách tiếp cận của các ngôn ngữ. Ví dụ Java compiler(javac) sau khi compile xong, JVM sẽ nhận input là java bytecodes từ javac, sau đó tiến hành các bước optimization bằng JIT(Just in time compiler) và translation qua machine code.

Còn với C, output của gcc sẽ là assembly code. Để chuyển hóa thành machine code cần thêm một utility tool gọi là assembler. (Với MS compiler thì trực tiếp build ra machine code luôn). Lí do để chỉ produce ra assembly code là để chia nhỏ công việc, dễ debug compiler. Nếu so với compilation thì việc translate từ assembly qua machine code khá đơn giản.

Frontend compiler

Như vậy là đã xong tổng qua compiler. Bây giờ mình sẽ đi vào phần đầu tiên – front end compiler.



Hình trên là bốn bước làm việc của một front end compiler. Mình sẽ đi từng vào từng phần.

Lexer

Nói về lexer chúng ta cần hiểu lexical analysic. Đây là một process để convert từ một chuỗi kí tự sang một chuỗi các “token”. Mỗi token sẽ có meaning riêng. Ví dụ khi bạn có một command:

`total = 5 + 6`

Với input vào dĩ nhiên là string, lexer sẽ giúp bạn categorize ở mức “word” như sau:

Lexeme	Token category
total	"Identifier"
=	"Assignment operator"
5	"Integer literal"
+	"Addition operator"
6	"Integer literal"
;	"End of statement"

Và đống token đã được categorized này, sẽ được đưa đến parser. Mình sẽ giới thiệu sau. Giờ chúng ta sẽ tiến hành implement thử trong Go.

Để implement được lexer, hầu hết các compiler đều sử dụng kiến trúc statemachine. Đại loại là luôn luôn có một cái machine đang chạy, xử lý xong cái này thì tới cái khác, đụng lỗi thì báo lỗi.
(Còn tiếp)