Module 01: CS31003: Compilers

Overview: Phases of a Compiler

Pralay Mitra
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

pralay@cse.iitkgp.ac.in
ppd@cse.iitkgp.ac.in

August 10, 2021

Course Outline

- Outline of Principles
- Outline of Implementation
- Books:
  - Compilers: Principles, Techniques, and Tools (2nd Edition) by A.V. Aho, Monica S Lam, R. Sethi, Jeffrey D. Ullman (Pearson / Addison-Wesley)
  - Flex and Bison by John Levine (O'Reilly)
  - Compiler Design in C by Allen Holub
  - Advanced Compiler Design and Implementation by Steven Muchnick
- Quora: **Is writing a compiler very easy?**
  It's something we teach undergrad CS majors to do in a semester. It might be the most code they've ever written that quickly (I recall mine took 10k lines in just five weeks) but most of them succeed. It might well be the most challenging program you write as an undergrad, but you should manage to go on to more difficult projects once you start your career. - Barry Rountree, Computer Scientist, LLNL:
  `https://www.quora.com/profile/Barry-Rountree`
  `https://www.quora.com/Is-writing-a-compiler-very-easy`

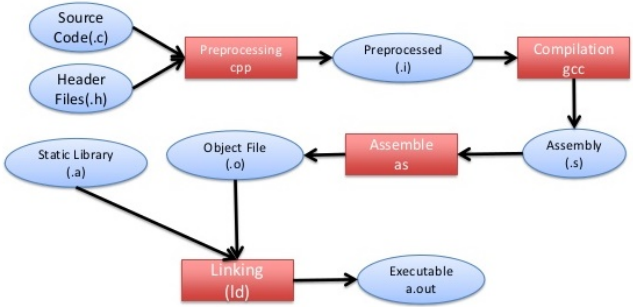
Module Objectives

- Understand an outline of the course
- Understand the phases of a compiler

Module Outline

Compiling a C Program

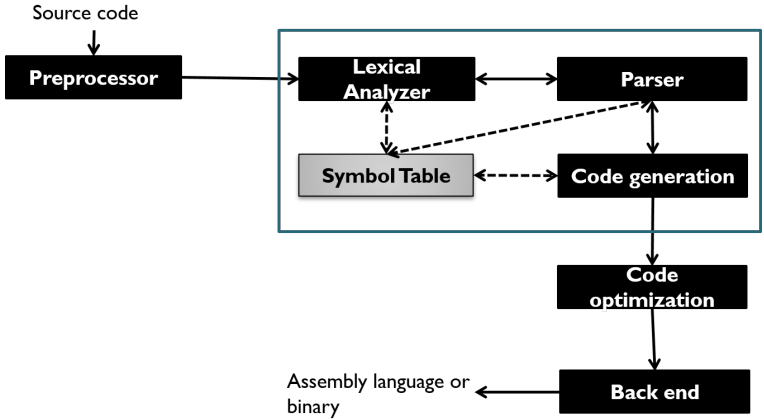
- C Pre-Processor (CPP)
- C Compiler
- Assembler
- Linker



Compilation Flow Diagrams for gcc

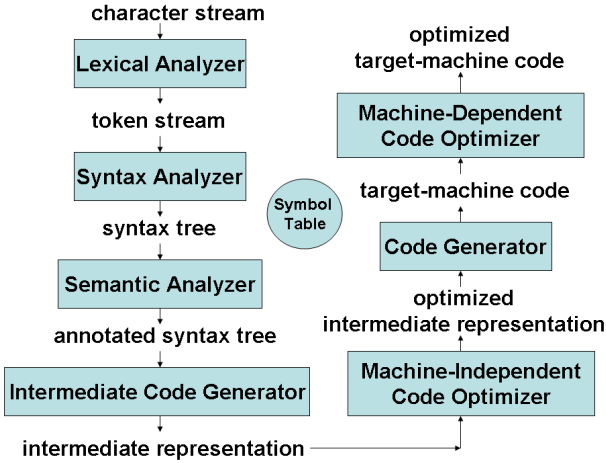
Source: *http://www.slideshare.net/Bletchley131/compilation-and-execution(slide#2)*

Compiling a C Program



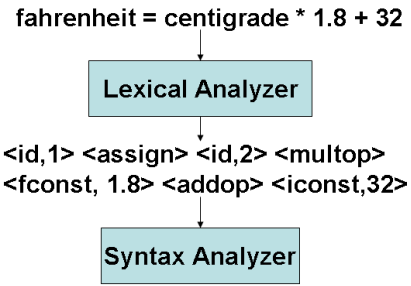
Four Pass Compiler

Phases



Source: Y N Srikant (NPTEL)

Lexical Analysis Phase



$$
\begin{aligned}
\textit{fahrenheit} &= \textit{centigrade} * 1.8 + 32 \\
\textit{totalAmount} &= \textit{principalAmount} * 10 + \textit{principalAmount} \\
\textit{finalVelocity} &= \textit{acceleration} * \textit{time} + \textit{initialVelocity}
\end{aligned}
$$

Source: Y N Srikant (NPTEL)

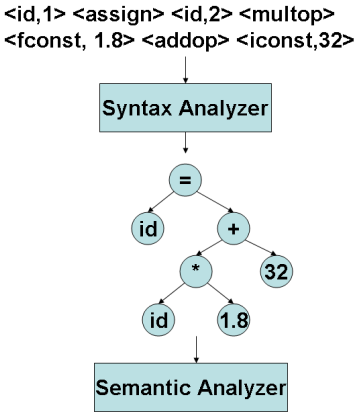
Lexical Analysis Phase

$$
\begin{aligned}
f &= c * 1.8 + 32 \\
b &= a * 10 + a \\
v &= a * t + u
\end{aligned}
$$

$$
\begin{aligned}
id &= id * num + num \\
id &= id * num + id \\
id &= id * id + id
\end{aligned}
$$

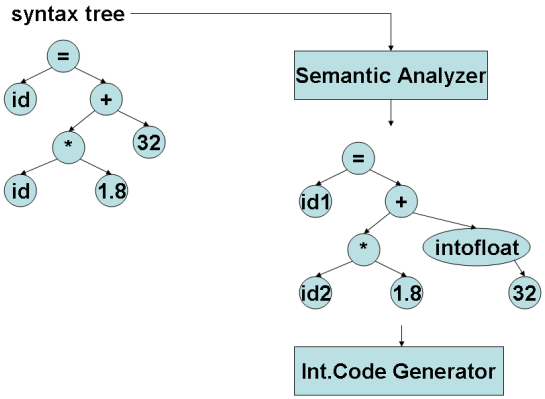
$$
\begin{aligned}
E &= E * E + E \\
(E &= ((E * E) + E))
\end{aligned}
$$

Syntax Analysis Phase



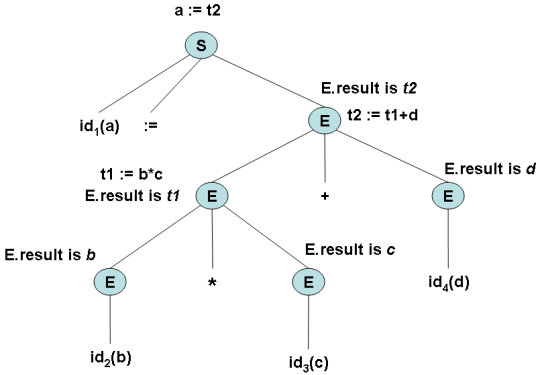
Source: Y N Srikant (NPTEL)

Semantic Analysis Phase



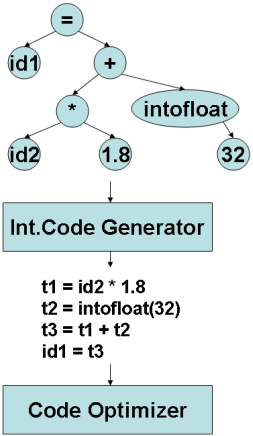
Source: Y N Srikant (NPTEL)

Expression Quads



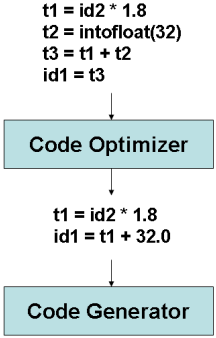
Source: Y N Srikant (NPTEL)

Intermediate Code Generator



Source: Y N Srikant (NPTEL)

Code Optimization



Source: Y N Srikant (NPTEL)

Code Generation and Optimization: Practice Example

* A+B*C+D

- t0=A
- t1=B
- t2=C
- t3=t1*t2
- t4=t0+t3
- t5=D
- t6=t4+t5

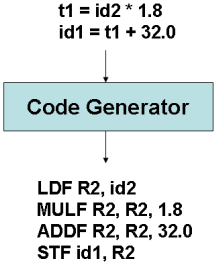
* t0=A
* t1=B
* t2=C
* t1=t1*t2
* t0=t0+t1
* t1=D
* t0=t0+t1

* t0=A
* t1=B
* t1=t1*C
* t1=t0+t1
* t1=t1+D

Target Code Generation

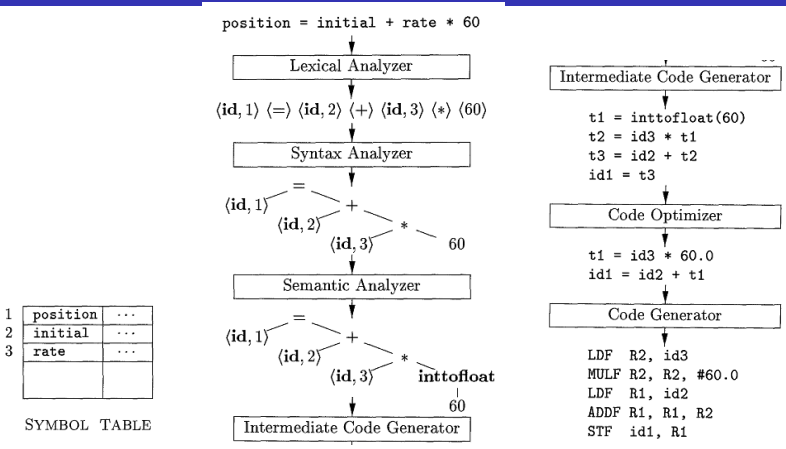
- Data Flow and Control Flow Analysis
- Registration Allocation and Assignment
- Code Generation

Target Code Generation



Source: Y N Srikant (NPTEL)

Sample pass through Phases



Source: Dragon Book

Figure: Translation of an assignment statement

Sample Translation


```
{
    int i; int j;
    float a[100]; float v; float x;

    while (true) {
        do i=i+1; while(a[i]<v);
        do j=j-1; while(a[j]>v);
        if (i>=j) break;
        x=a[i]; a[i]=a[j]; a[j]=x;
    }
}
```

```
01: i = i + 1
02: t1 = a [ i ]
03: if t1 < v  goto 01
04: j = j - 1
05: t2 = a [ j ]
06: if t2 > v  goto 04
07: ifFalse i >= j goto 09
08: goto 14
09: x = a [ i ]
10: t3 = a [ j ]
11: a [ i ] = t3
12: a [ j ] = x
13: goto 01
14: .
```

A Typical Compiler Techniques

Promote high level languages by minimizing the execution overhead

Support HPC systems

Compiler

Support several source languages

Support several target machines

Potential to translate correctly infinite set of programs written in the source language.

Collection of compilers

Software engineering techniques

Generate optimal target code from source program ??

Languages by Translation Types

Language	Compilation	Typing	Framework
C	Static	Weak[1], Static	No
C++	Static	Strong[2], Static[3]	No[4]
Java	Static	Strong, Static[5]	Yes[6]
Python	Dynamic[7]	Strong, Dynamic	Yes[8]

[1] For example, `void*` breaking typing
[2] If typical C features are not used
[3] Dynamic w/ Polymorphism
[4] RTTI for `dynamic_cast`
[5] Dynamic w/ Polymorphism
[6] Java Virtual Machine – JVM
[7] Interpreter
[8] Python Virtual Machine – PVM

Module Summary

- Outline of Course and Material provided
- Recap on the outline of C Compilation Process
- Brief discussion on Phases of a Compiler to understand
  - Front-end flow: Language to TAC
  - Back-end flow: TAC to Machine
- Infix to Postfix Translation
- Outline of languages with translation types