CS 348: Module 02: Compilers

Overview: Phases of a Compiler

Prof. Sukumar Nandi

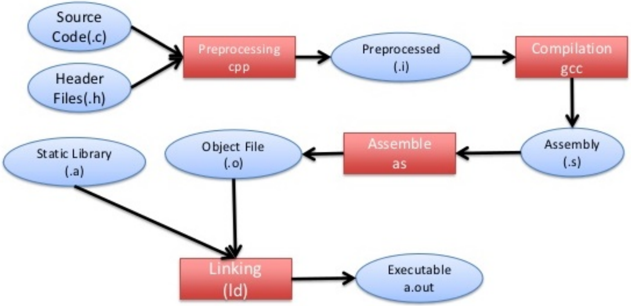
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

sukumar@iitg.ac.in

February 4, 2025

Compiling a C Program

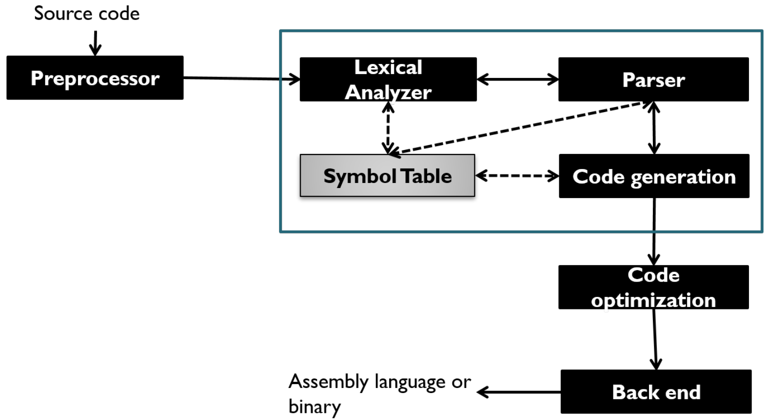
- C Pre-Processor (CPP)
- C Compiler
- Assembler
- Linker



Compilation Flow Diagrams for gcc

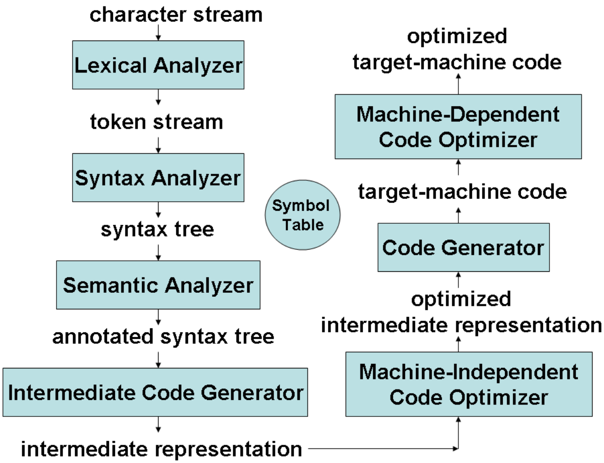
Source: *http://www.slideshare.net/Bletchley131/compilation-and-execution(slide#2)*

Compiling a C Program



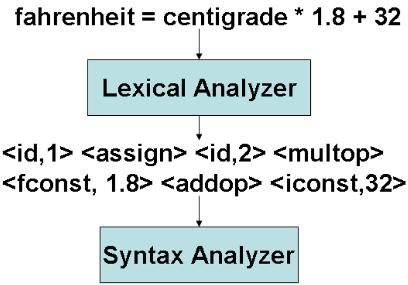
Four Pass Compiler

Phases


character stream
Lexical Analyzer
token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
annotated syntax tree
Intermediate Code Generator
intermediate representation
Machine-Independent Code Optimizer
optimized intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
optimized target-machine code
Symbol Table

Source: Y N Srikant (NPTEL)

Lexical Analysis Phase



fahrenheit = centigrade * 1.8 + 32

<id,1> <assign> <id,2> <multop>
<fconst, 1.8> <addop> <iconst,32>

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

Source: Y N Srikant (NPTEL)

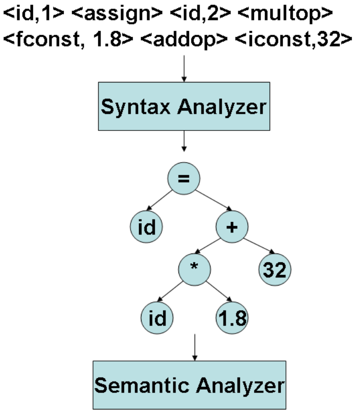
Lexical Analysis Phase

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

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

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

Syntax Analysis Phase



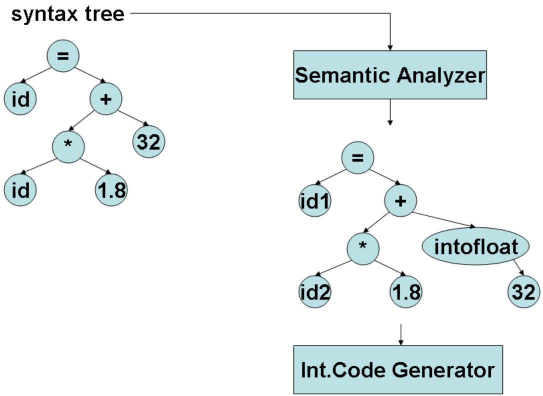
<id,1> <assign> <id,2> <multop> <fconst, 1.8> <addop> <iconst,32>

Syntax Analyzer

Semantic Analyzer

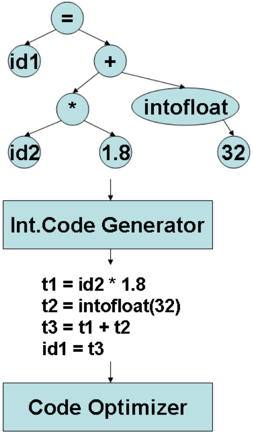
Source: Y N Srikant (NPTEL)

Semantic Analysis Phase



Source: Y N Srikant (NPTEL)

Intermediate Code Generator



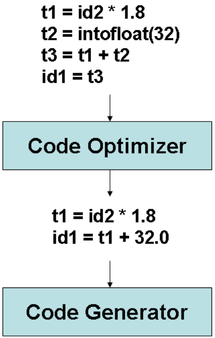
Int.Code Generator

t1 = id2 * 1.8
t2 = intofloat(32)
t3 = t1 + t2
id1 = t3

Code Optimizer

Source: Y N Srikant (NPTEL)

Code Optimization



```
t1 = id2 * 1.8
t2 = intofloat(32)
t3 = t1 + t2
id1 = t3
```

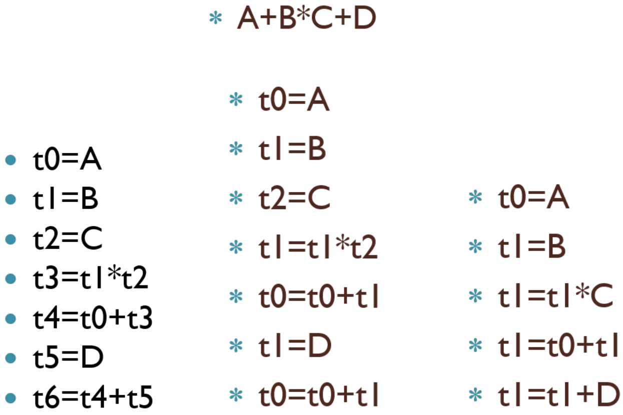
Code Optimizer

```
t1 = id2 * 1.8
id1 = t1 + 32.0
```

Code Generator

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

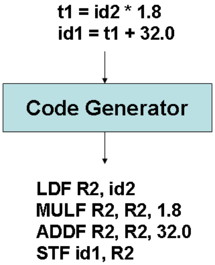
- t0=A
- t1=B
- t2=C
- t1=t1*t2
- t0=t0+t1
- t1=D
- t0=t0+t1

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

Target Code Generation

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

Target Code Generation



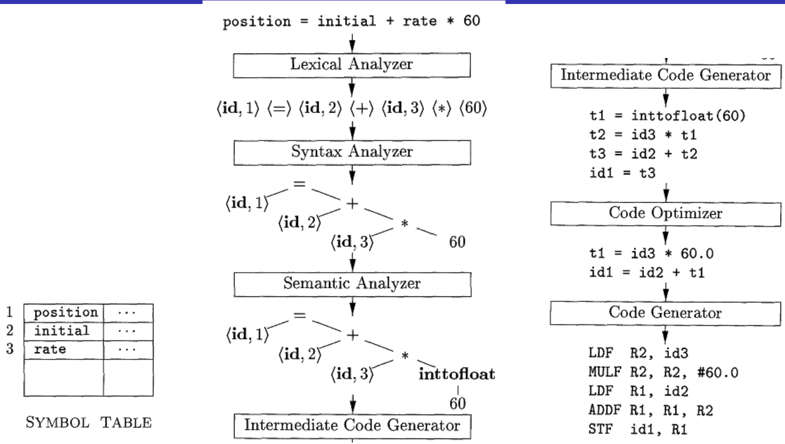
```
t1 = id2 * 1.8
id1 = t1 + 32.0
```

Code Generator

```
LDF R2, id2
MULF R2, R2, 1.8
ADDF R2, R2, 32.0
STF id1, R2
```

Source: Y N Srikant (NPTEL)

Sample pass through Phases



```
position = initial + rate * 60
```

Lexical Analyzer

$\langle \mathbf{id}, 1\rangle\ \langle=\rangle\ \langle \mathbf{id}, 2\rangle\ \langle+\rangle\ \langle \mathbf{id}, 3\rangle\ \langle*\rangle\ \langle 60\rangle$

Syntax Analyzer

Semantic Analyzer

Intermediate Code Generator

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Code Optimizer

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Code Generator

```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

1	position	···
2	initial	···
3	rate	···

SYMBOL TABLE

Source: Dragon Book

Figure: Translation of an assignment statement

A Typical Compiler Techniques

Promote high level languages by minimizing the execution overhead

Support HPC systems

Compiler

Support several source languages

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

Support several target machines

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

- 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
- Outline of languages with translation types