Compiler (Introduction)

زینب زالی
دانشگاه صنعتی اصفهان

Why we study compilers

Why we study Compilers

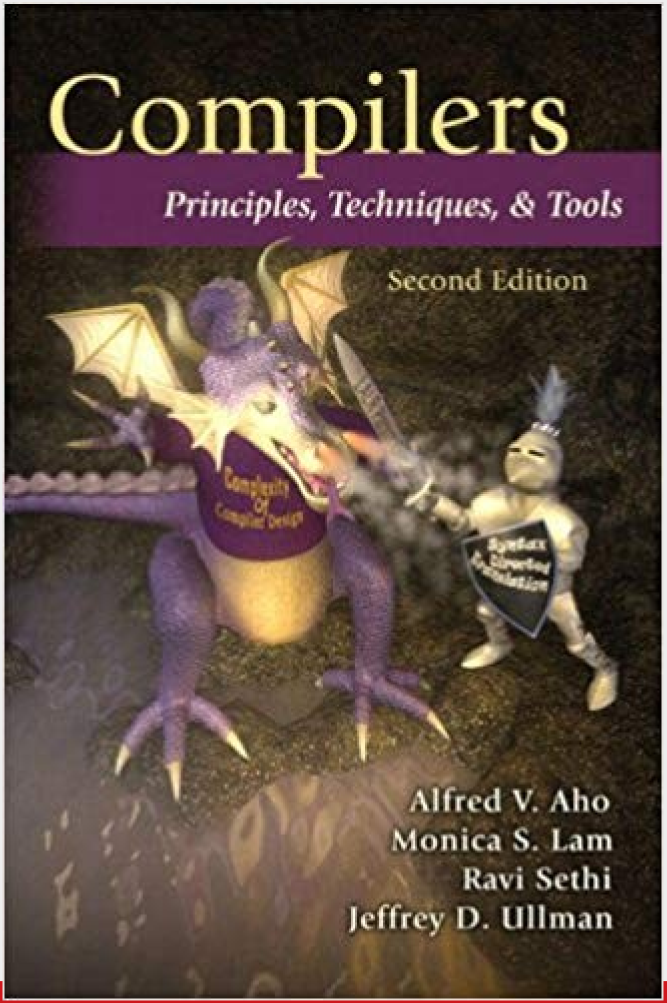
- may be you invent a very different general purpose language like python
- some examples that use Compiler theory and techniques:
  - Natural languages translators, XML, Json, Grep
- A domain-specific language is required
  - Ex: P4 in networking, Latex
- It is a good exercise for the brain
- It is a very good exercise for programming

Compiler science

- The first compiler had a huge impact on computer science
- Led to enormous body of theoretical work

Reference

- Dragon book


Compilers
Principles, Techniques, & Tools
Second Edition
Complexity of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

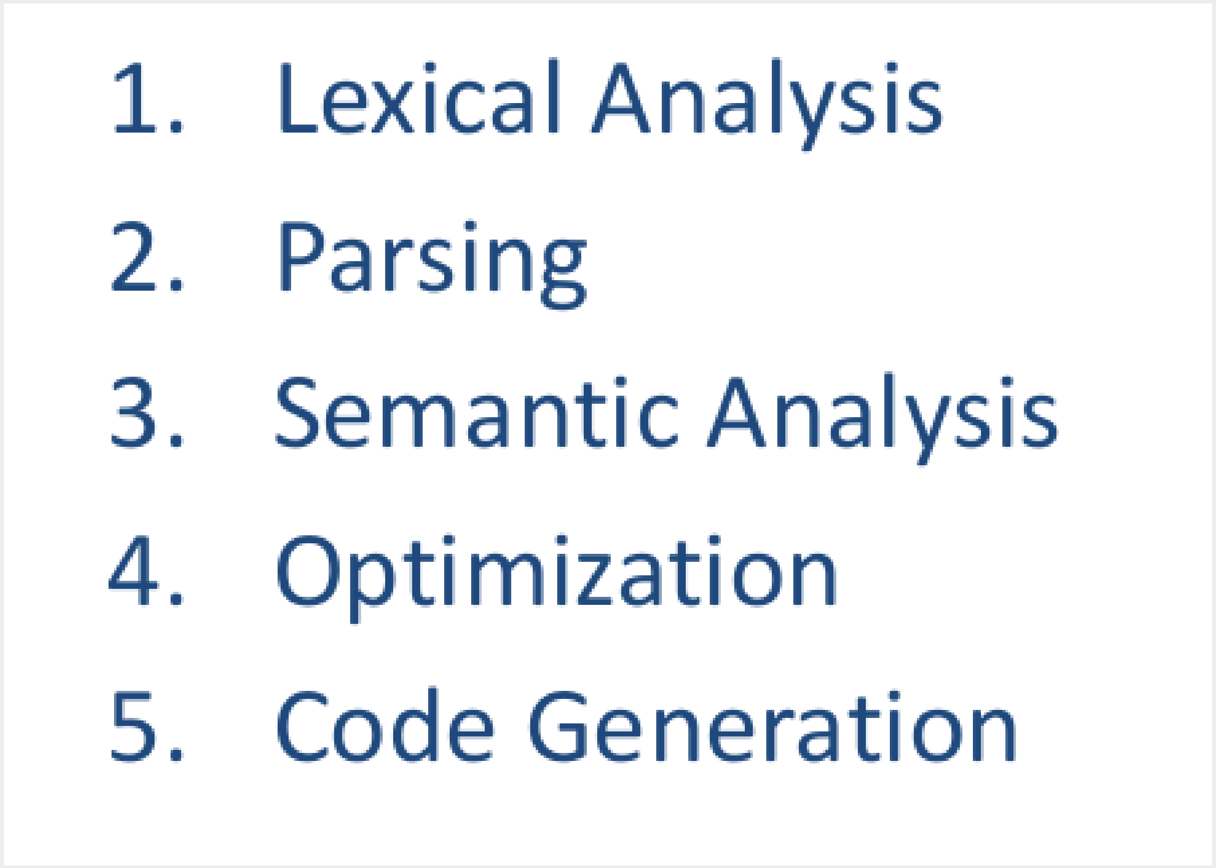
Grading

Quiz Class activities	2 1
Midterm	6
Final	6
Theoretical HW Programming HW	2 4 to 5

Some Notes:
Generally No Book is required
Slides
White board

- how do you think about compiler functions and its steps?

Phases of a Compilers



1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

John Backus

Speedcoding
(1953)

Interpreter

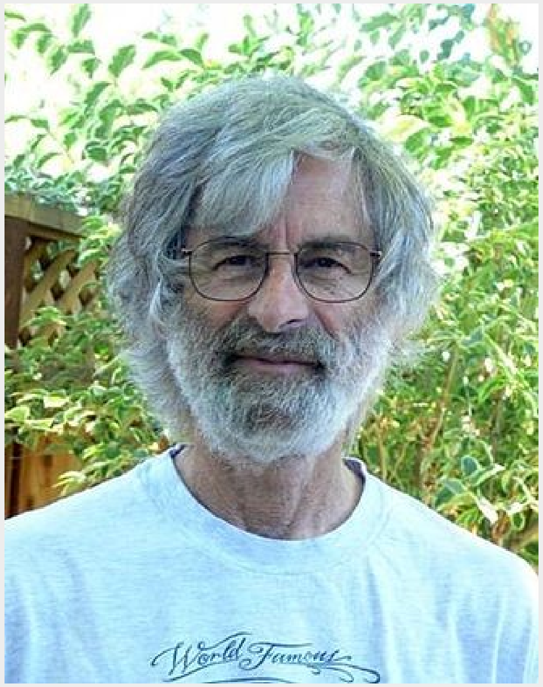
Fortran
(1954)

First high level
programming

BNF
(Normal Form)

Leslie Lamport

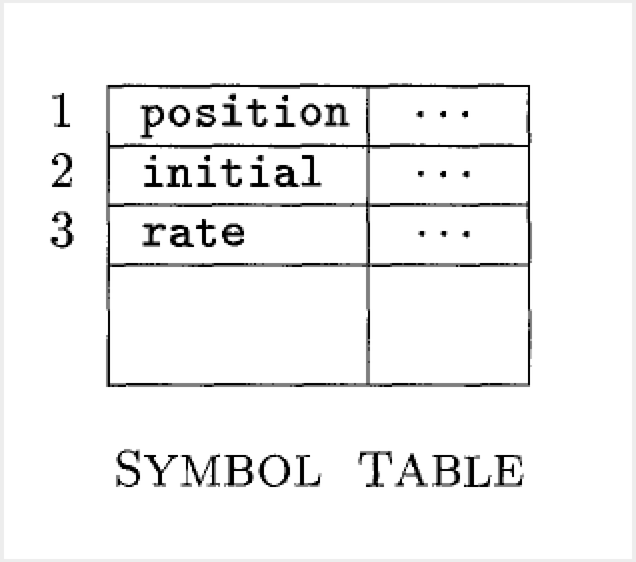
Distributed algorithms



Latex

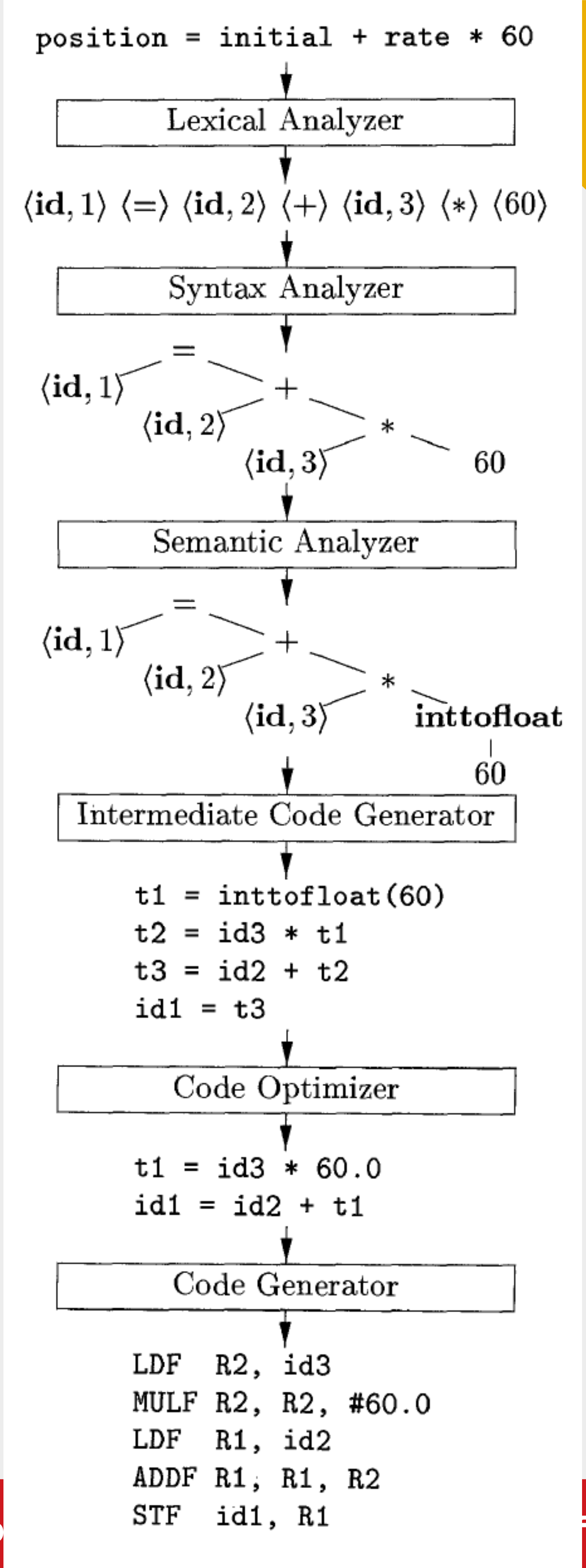
Which programming language for writing a compiler?

- No matter which language the compiler is written with. The compiler is executed through running the final machine code of the compiler
  - So new version of gcc can be written with the old ones (self-hosting)
- C (Denis Richie) is written by B.



1	position	...
2	initial	...
3	rate	...

SYMBOL TABLE



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

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

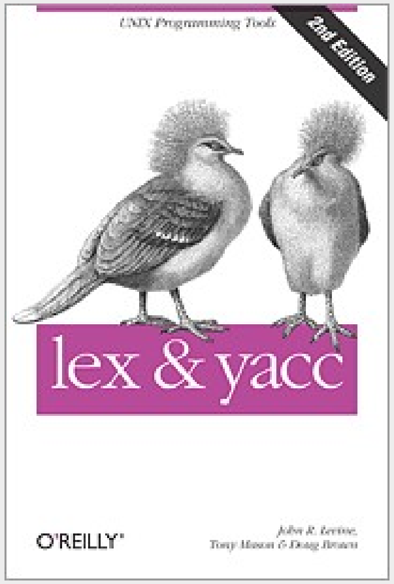
```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

Lex (Flex), Yacc (Bison)

Lex (Flex): A lexical analyzer generator using Regular experesions

Yacc (Bison): A tools for parse a language which is described by a context-free LALR(1) grammar.

llvm: the LLVM Project is a collection of modular and reusable compiler and toolchain technologies



Plan

	Subject	HW/ Project
1	Introduction Lexical Analysis	
2	Lexical Analysis Lexical Analysis (DFA NFA complexity)	
3	Lexical Analysis (RE to DFA) Syntax Analysis (introducing Top-down)	P1 (start)
4	Syntax Analysis (Recursive Descent) Syntax Analysis (disambiguating, left recursion, left factoring)	
5	Syntax Analysis (predictive recursive descent) Syntax Analysis (LL1 grammars, first, follow)	
6	Syntax Analysis (LL1 parsing) Syntax Analysis (LL1 error handling)	
7	Syntax Analysis (Bottom up parsing) Syntax Analysis (Shift reduce, handle)	
8	Syntax Analysis (Viable prefix) Syntax Analysis (LR0 automata)	

Plan

	Subject	HW/ Project
9	Syntax Analysis (LR parser, SLR) Syntax Analysis (LR parser, SLR)	
10	Syntax Analysis (CLR) Syntax Analysis (LALR)	
11	Syntax Analysis(LR ambiguate languages, error handling) Semantic Analysis(Syntax Directed Translation, SDD)	P2 (start)
12	Semantic Analysis(SDD calculation,L-attibuted grammar) Semantic Analysis(AST using SDD)	
13	Semantic Analysis(SDD implementation) Semantic Analysis(SDT scheme)	P3 (start)
14	Type checking	
15	Code generation	
16	Optimization	