Lecture 01 - Introduction

THEORY OF COMPILATION

Eran Yahav

1

Who?

Eran Yahav
Taub 734
Tel: 8294318
yahave@cs.technion.ac.il
Monday 13:30-14:30
http://www.cs.tecnion.ac.il/~yahave

2

What?

- Understand
  - what is a compiler
  - how does it work
  - techniques that can be re-used in other settings
- What will help us
  - Text books
    - Modern compiler design
    - Compilers: principles, techniques and tools
  - 5 homework assignments
- Will also help
  - Taking a deep breath
  - Focusing on material and not on your grade

3

What is a Compiler?

- "A compiler is a computer program that transforms source code written in a programming language (source language) into another language (target language). The most common reason for wanting to transform source code is to create an executable program."

--Wikipedia

4

What is a Compiler?

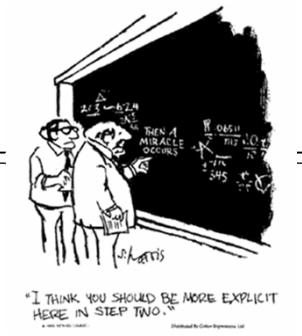
source language

C
C++
Pascal
Java
...
Postscript
TeX
Perl
JavaScript
Python
Ruby

Prolog

Lisp
Scheme
ML
OCaml

txt
Source text



Compiler

target language

IA32
IA64
SPARC

C
C++
Pascal
Java

exe
Executable code

Java Bytecode
...

5

What is a Compiler?

Compiler

txt
Source text

exe
Executable code

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,2
SAL R1
INC R1
MOV R2,R1

6

Anatomy of a Compiler

Compiler

txt
Source text

Frontend (analysis)

Semantic Representation

Backend (synthesis)

exe
Executable code

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,2
SAL R1
INC R1
MOV R2,R1

7

Modularity

SET R1,2
STORE #0,R1
SHIFT R1,1
STORE #1,R1
ADD R1,1
STORE #2,R1

txt
Source Language 1

Frontend SL1
Frontend SL2
Frontend SL3

Semantic Representation

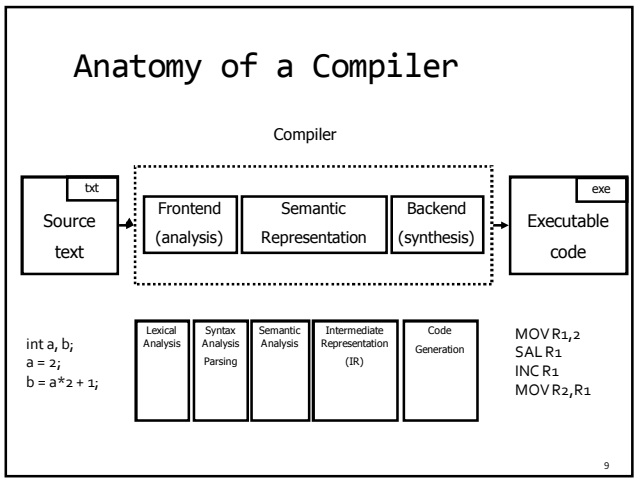
Backend TL1
Backend TL2
Backend TL3

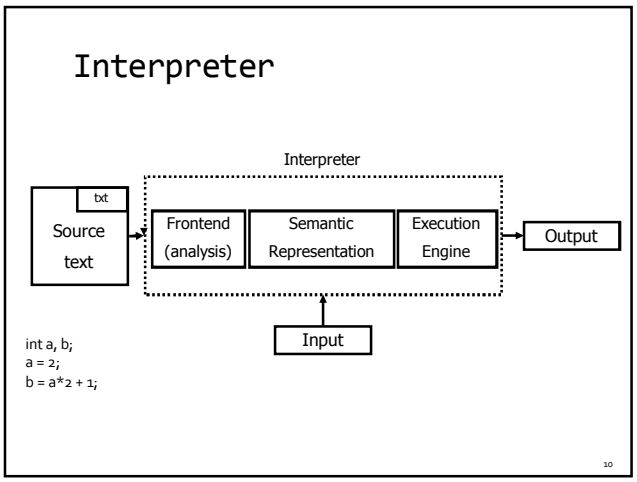
exe
Executable target 1

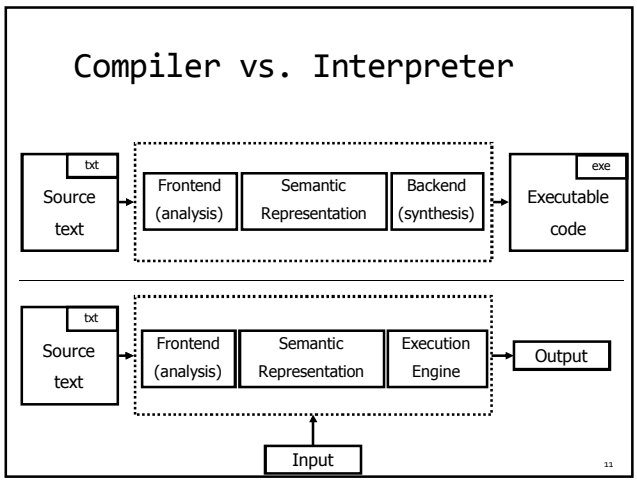
int a, b;
a = 2;
b = a*2 + 1;

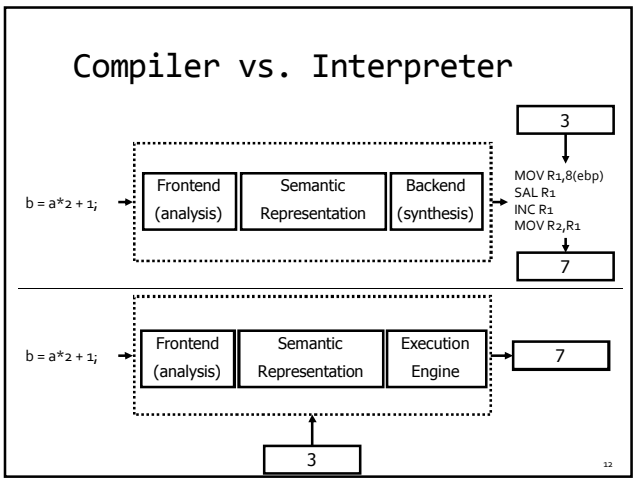
MOV R1,2
SAL R1
INC R1
MOV R2,R1

8

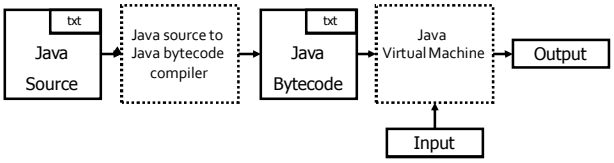

Anatomy of a Compiler
Compiler
txt
Source text
Frontend (analysis)
Semantic Representation
Backend (synthesis)
exe
Executable code
int a, b;
a = 2;
b = a*2 + 1;
Lexical Analysis
Syntax Analysis Parsing
Semantic Analysis
Intermediate Representation (IR)
Code Generation
MOV R1,2
SAL R1
INC R1
MOV R2,R1
9


Interpreter
Interpreter
txt
Source text
Frontend (analysis)
Semantic Representation
Execution Engine
Output
Input
int a, b;
a = 2;
b = a*2 + 1;
10


Compiler vs. Interpreter
txt
Source text
Frontend (analysis)
Semantic Representation
Backend (synthesis)
exe
Executable code
txt
Source text
Frontend (analysis)
Semantic Representation
Execution Engine
Output
Input
11


Compiler vs. Interpreter
b = a*2 + 1;
Frontend (analysis)
Semantic Representation
Backend (synthesis)
3
MOV R1,8(ebp)
SAL R1
INC R1
MOV R2,R1
7
b = a*2 + 1;
Frontend (analysis)
Semantic Representation
Execution Engine
7
3
12

Just-in-time Compiler (Java example)



Just-in-time compilation: bytecode interpreter (in the JVM) compiles program fragments during interpretation to avoid expensive re-interpretation.

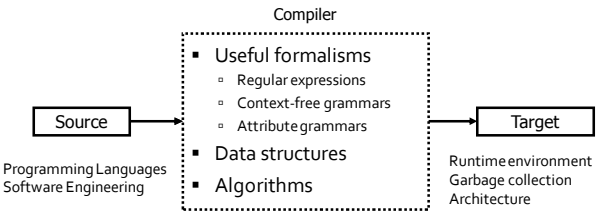
13

Why should you care?

- Every person in this class will build a parser some day
  - Or wish he knew how to build one…
- Useful techniques and algorithms
  - Lexical analysis / parsing
  - Semantic representation
  - …
  - Register allocation
- Understand programming languages better
- Understand internals of compilers
- Understand (some) details of target architectures

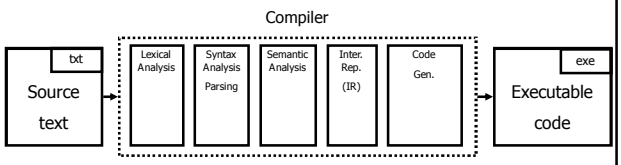
14

Why should you care?



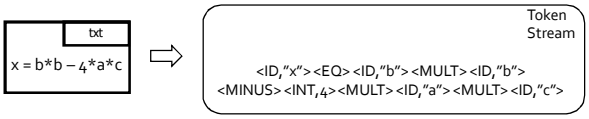
15

Course Overview



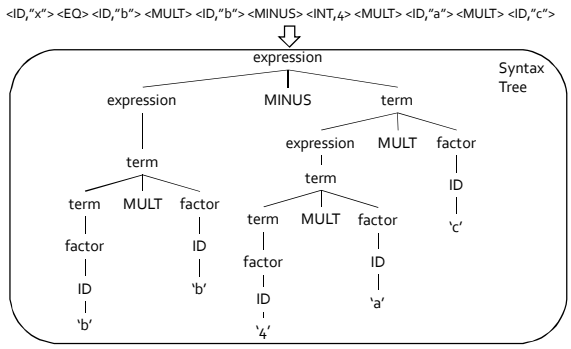
16

Journey inside a compiler



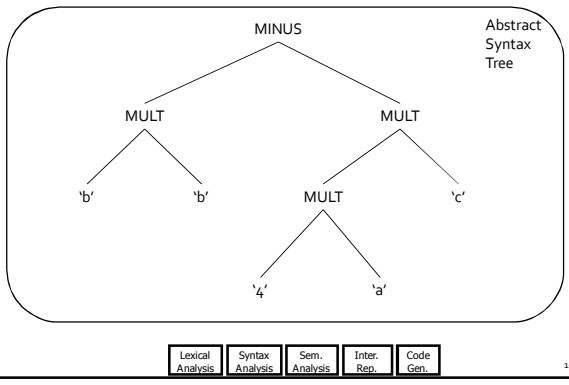
17

Journey inside a compiler



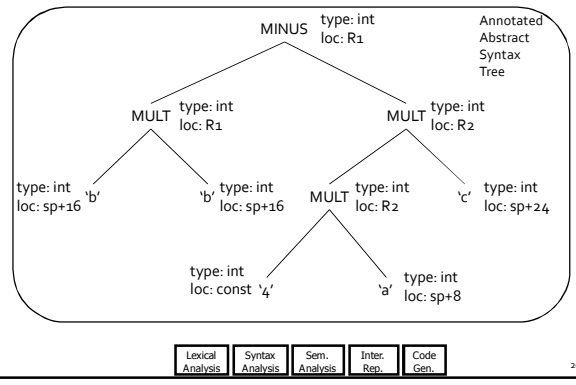
18

Journey inside a compiler



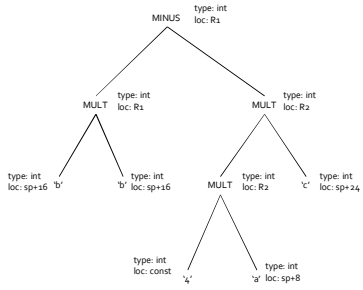
19

Journey inside a compiler



20

Journey inside a compiler

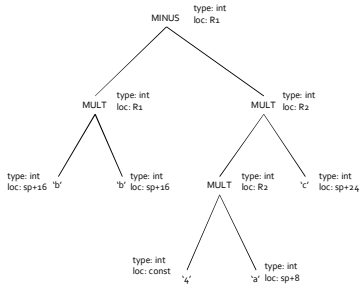


Intermediate Representation

R2 = 4*a
R1=b*b
R2= R2*c
R1=R1-R2

Lexical Analysis | Syntax Analysis | Sem. Analysis | Inter. Rep. | Code Gen.

Journey inside a compiler



Intermediate Representation

R2 = 4*a
R1=b*b
R2= R2*c
R1=R1-R2

Assembly Code

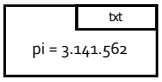
MOV R2,(sp+8)
SAL R2,2
MOV R1,(sp+16)
MUL R1,(sp+16)
MUL R2,(sp+24)
SUB R1,R2

Lexical Analysis | Syntax Analysis | Sem. Analysis | Inter. Rep. | Code Gen.

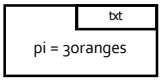
Error Checking

- In every stage…
- Lexical analysis: illegal tokens
- Syntax analysis: illegal syntax
- Semantic analysis: incompatible types, undefined variables, …
- Every phase tries to recover and proceed with compilation (why?)
  - Divergence is a challenge

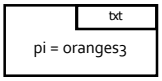
Errors in lexical analysis



⇨ Illegal token

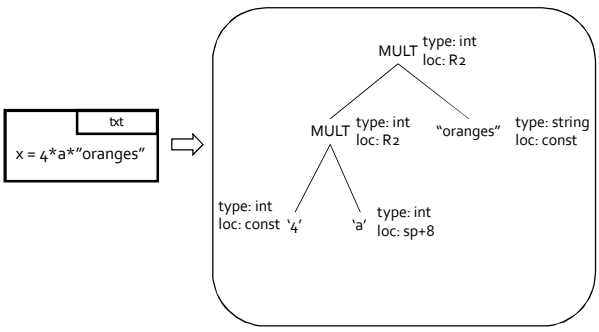


⇨ Illegal token



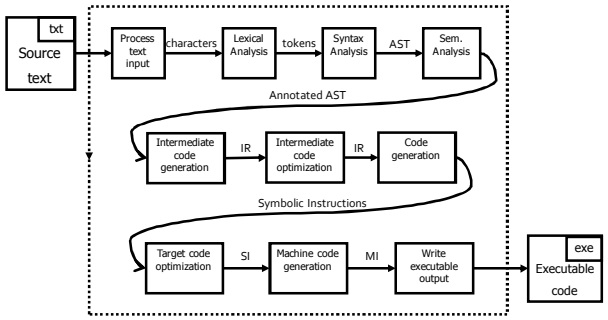
⇨ <ID,"pi">, <EQ>, <ID,"oranges3">

Error detection: type checking



25

The Real Anatomy of a Compiler



26

Optimizations

- "Optimal code" is out of reach
  - many problems are undecidable or too expensive (NP-complete)
  - Use approximation and/or heuristics
- Loop optimizations: hoisting, unrolling, …
- Peephole optimizations
- Constant propagation
  - Leverage compile-time information to save work at runtime (pre-computation)
- Dead code elimination
  - space
- …

27

Machine code generation

- Register allocation
  - Optimal register assignment is NP-Complete
  - In practice, known heuristics perform well
- assign variables to memory locations
- Instruction selection
  - Convert IR to actual machine instructions
- Modern architectures
  - Multicores
  - Challenging memory hierarchies

28

Compiler Construction Toolset

- Lexical analysis generators
  - lex
- Parser generators
  - yacc
- Syntax-directed translators
- Dataflow analysis engines

29

Summary

- Compiler is a program that translates code from source language to target language
- Compilers play a critical role
  - Bridge from programming languages to the machine
  - Many useful techniques and algorithms
  - Many useful tools (e.g., lexer/parser generators)
- Compiler constructed from modular phases
  - Reusable
  - Different front/back ends

30

Coming up next

- Lexical analysis

31