CS 153: Concepts of Compiler Design

August 24 Class Meeting

Department of Computer Science
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

Basic Info

- Office hours
  - TuTh 3:00 – 4:00 PM
  - ENG 250
- Website
  - Faculty webpage: http://www.cs.sjsu.edu/~mak/
  - Class webpage: http://www.cs.sjsu.edu/~mak/CS153/index.html
  - Syllabus
  - Assignments
  - Lecture notes

Permission Codes?

- If you need a permission code to enroll in this class, please fill out and hand in a filled-out and signed “Add Code Information” form.
  - Be sure to list prerequisite courses that you’ve successfully completed.
  - Prerequisites: CS 47 or CMPE 102, CS 146, and CS 154 (with a grade of "C-" or better in each); Computer Science, Applied and Computational Math, or Software Engineering majors only.
- Priority will be given to graduating seniors.
  - You must show your graduating senior card.

Goals of the Course

- Understand the concepts of compilers and interpreters.
  - Parser, scanner, tokens
  - Symbol tables, intermediate code
  - Executors, code generators
  - Compiler-compilers

© R. Mak

Goals of the Course, *cont'd*

- Learn important job skills that employers want!
- Work on a small programming team.
- Understand and modify a Big Hairy Legacy Application.
- Use modern software engineering practices to develop a complex application.

Course Overview

- **First half:** Modify a Pascal interpreter.
  - The interpreter is written in Java (the implementation language).
  - The source programs are written in Pascal (the source language).
  - The implementation code for the interpreter will be presented to you incrementally.
- *Midterm*

Course Overview, *cont'd*

- **Second half:** Your compiler project.
  - ANTLR 4 compiler-compiler
  - Java Virtual Machine (JVM) architecture
  - Jasmin assembly language
  - Back end code generator

- *Final*

Required Textbooks

- *Writing Compilers and Interpreters,* 3rd edition
  - Author: Ronald Mak
  - Publisher: Wiley
  - ISBN: 978-0-470-17707-5
  - Source code: http://www.cs.sjsu.edu/~mak/CS153/sources/ (Java and C++)

Use during the entire semester.

- *The Definitive ANTLR 4 Reference*
  - Author: Terence Parr
  - Publisher: Pragmatic Bookshelf
  - ISBN: 978-1934356999
  - URL: http://www.antlr.org

Use during the second half of the semester.

Project Teams

- Projects will be done by small project teams.
  - Projects will be broken up into assignments.
- Form your own teams of 4 members each.
- Choose your team members wisely!
  - Be sure you'll be able to meet and communicate with each other and work together well.
  - No moving from team to team.
- Each team member will receive the same score on each team assignment and team project.

Project Teams, *cont'd*

- Each team email to ron.mak@sjsu.edu by Wednesday, August 30:
  - Your team name
  - A list of team members and email addresses
- Subject: **`CS 153 Team`** *Team Name*
  - Example: **`CS 153 Team Hyper Hackers`**

Individual Responsibilities

You are personally responsible for participating and contributing to your team's work, and for understanding each part of the work for every assignment whether or not you worked on that part.

Postmortem Assessment Report

- At the end of the semester, each student will individually turn in a short (one page) report:
  - A brief description of what you learned in the course.
  - An assessment of your personal accomplishments for your project team.
  - An assessment of the contributions of each of your project team members.
- This report will be seen only by the instructor.

Your Individual Overall Class Grade

- 30% assignments
- 35% project
- 15% midterm
- 20% final

Your final class grade will be adjusted up or down depending on your level and quality of participation, as reported by your teammates' postmortem reports.

- During the semester, keep track of your progress in Canvas.
- At the end of the semester, students with the median score will get the B grade.
- Higher and lower grades will then be assigned based on how the scores cluster above and below the median.
- Therefore, your final class grade will be based primarily on your performance relative to the other students in the class.

Participation is Important

- Can move your final grade up or down, especially in borderline cases.
- Participation in class.
- Participation in your team.
  - As reported by the postmortem assessment reports.

Take roll!

Compiler Magic?

C compiler:

```
int main()
{
    printf("Hello, C world!\n");
}
```

Pascal compiler:

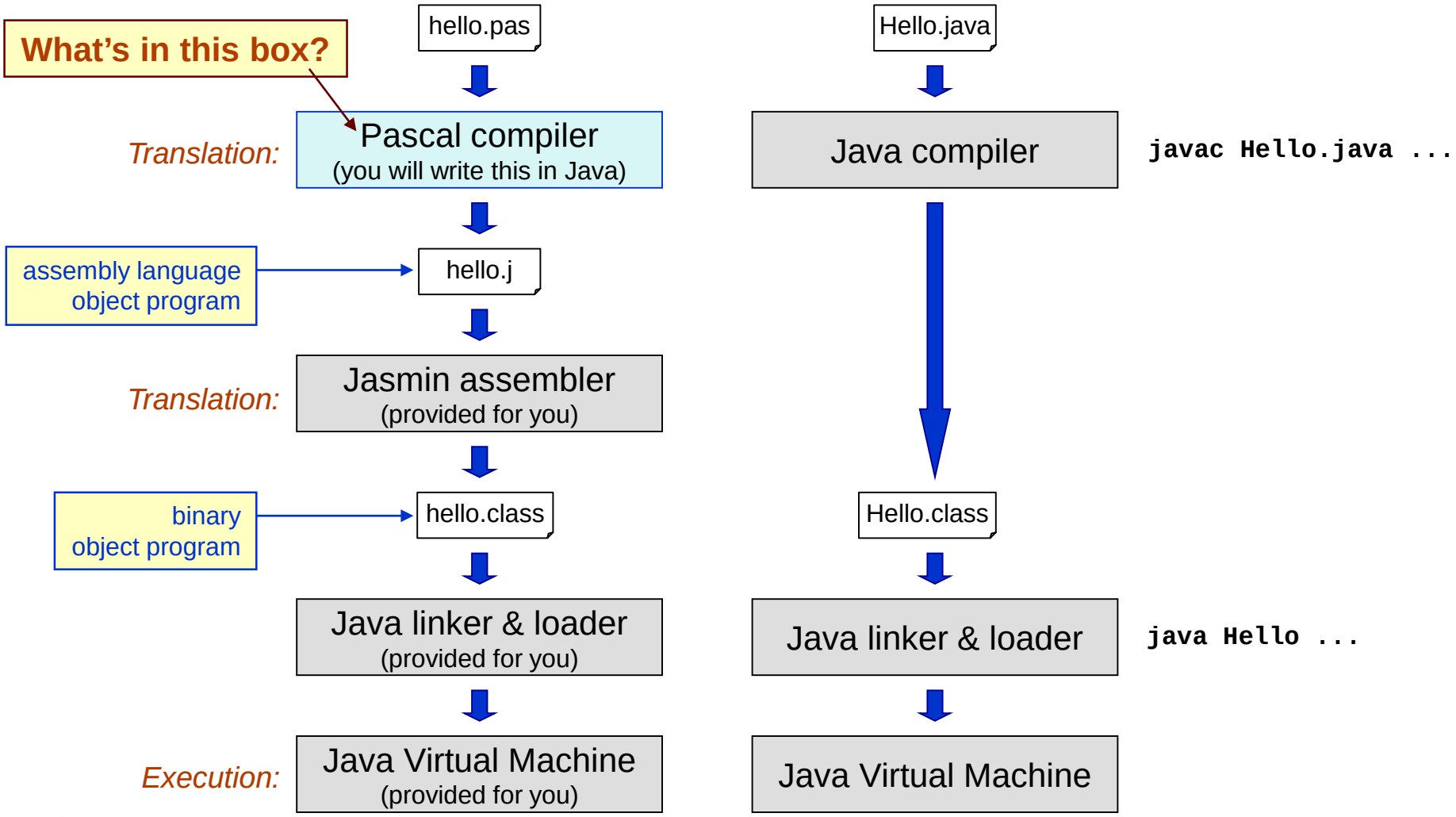
```
PROGRAM hello;

BEGIN
    writeln('Hello, Pascal world!');
END.
```

Java compiler:

```
public class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello, Java world!");
    }
}
```


© R. Mak

Overview of the Compilation Process



What is a Compiler?

- A software utility that is extremely important for developing applications …
- … usually overlooked and taken for granted ...
- *UNLESS* you can't get your program to compile!

A Compiler is a Translator

- A compiler translates a program that you've written
- ... in a high-level language
  - C, C++, Java, Pascal, etc.
- ... into a low-level language
  - assembly language or machine language
- ... that a computer can understand and eventually execute.

Assignment #1

- Posted to the class web page: http://www.cs.sjsu.edu/~mak/CS153/index.html
- Write a simple Pascal program.
- An individual (not team) assignment.
  - You may work together to learn the language.
  - But what you turn in must be your own work.
- Due Friday, September 1.

Assignment #1, *cont'd*

- Download and install the Lazarus/Free Pascal IDE for Pascal: https://www.lazarus-ide.org
- Online Pascal tutorials:
  - https://www.tutorialspoint.com/pascal/
  - http://www.taoyue.com/tutorials/pascal/

Assignment #1, *cont'd*

- Sample Pascal program: http://www.cs.sjsu.edu/~mak/CS153/assignments/1/EmployeeListing.pas
  - Input file: http://www.cs.sjsu.edu/~mak/CS153/assignments/1/employees.txt

More Definitions

- source program: the program (application) that you write in a high-level language which the compiler will translate
  - Usually stored in a source file.

- source language: the high-level language in which you write your source program
  - Example: Pascal

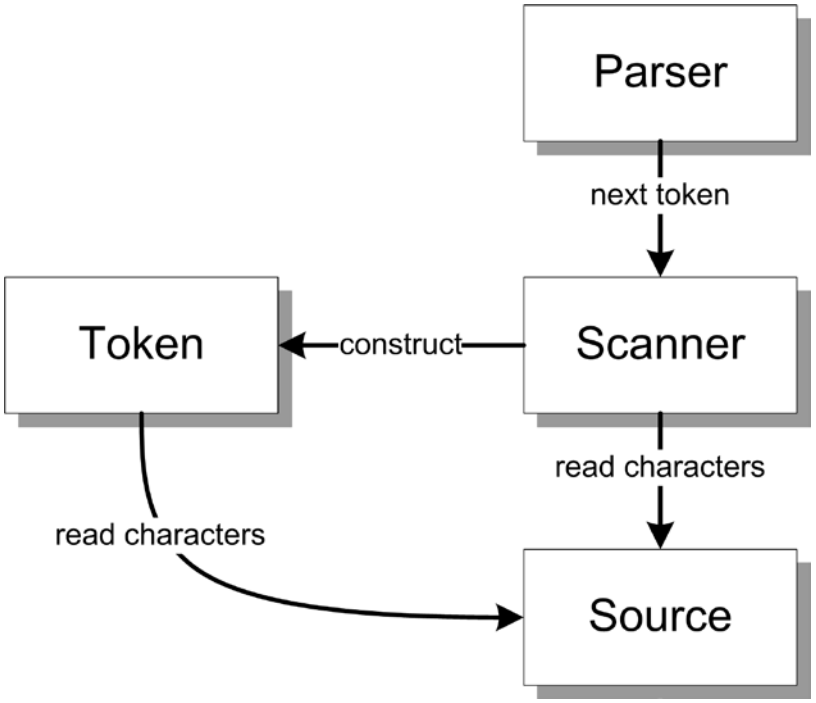
More Definitions, *cont'd*

- object language: the low-level language (AKA target language) into which the compiler translates the source program
  - Do not confuse *object language* with *object-oriented language*.
  - Example: Jasmin assembly language
  - Example: Intel machine code
- object program: your program after it has been translated into the object language

More Definitions, *cont'd*

- target machine: the computer that will eventually execute the object program
  - Example: Your laptop's hardware
  - Example: The Java Virtual Machine (JVM)
    - The JVM runs on your workstation or laptop (any computer that supports Java)
- implementation language: the language that the compiler itself is written in
  - Example: Java

Conceptual Design (Version 1)



- Parser
  - Controls the translation process.
  - Repeatedly asks the scanner for the next token.
- Scanner
  - Repeatedly reads characters from the source to construct tokens for the parser.
- Token
  - A source language element
    - identifier (name)
    - number
    - special symbol (+ - * / = etc.)
    - reserved word
  - Also reads from the source
- Source
  - The source program

Token

- A low-level element of the source language.
  - AKA lexeme
- Pascal language tokens
  - Identifiers
    - names of variables, types, procedures, functions, enumeration values, etc.
  - Numbers
    - integer and real (floating-point)
  - Reserved words
    - **`BEGIN END IF THEN ELSE AND OR NOT`** etc.
  - Special symbols
    - **`+ - * / := < <= = >= > . , .. : ( ) [ ] { } '`**

© R. Mak

Parser

- Controls the translation process.
  - Repeatedly asks the scanner for the next token.
- Knows the syntax ("grammar") of the source language's statements and expressions.
  - Analyzes the sequence of tokens to determine what kind of statement or expression it is translating.
  - Verifies that what it's seeing is syntactically correct.
  - Flags any syntax errors that it finds and attempts to recover from them.

Parser, *cont'd*

- What the parser does is called parsing.
  - It *parses* the source program in order to translate it.
  - AKA syntax analyzer

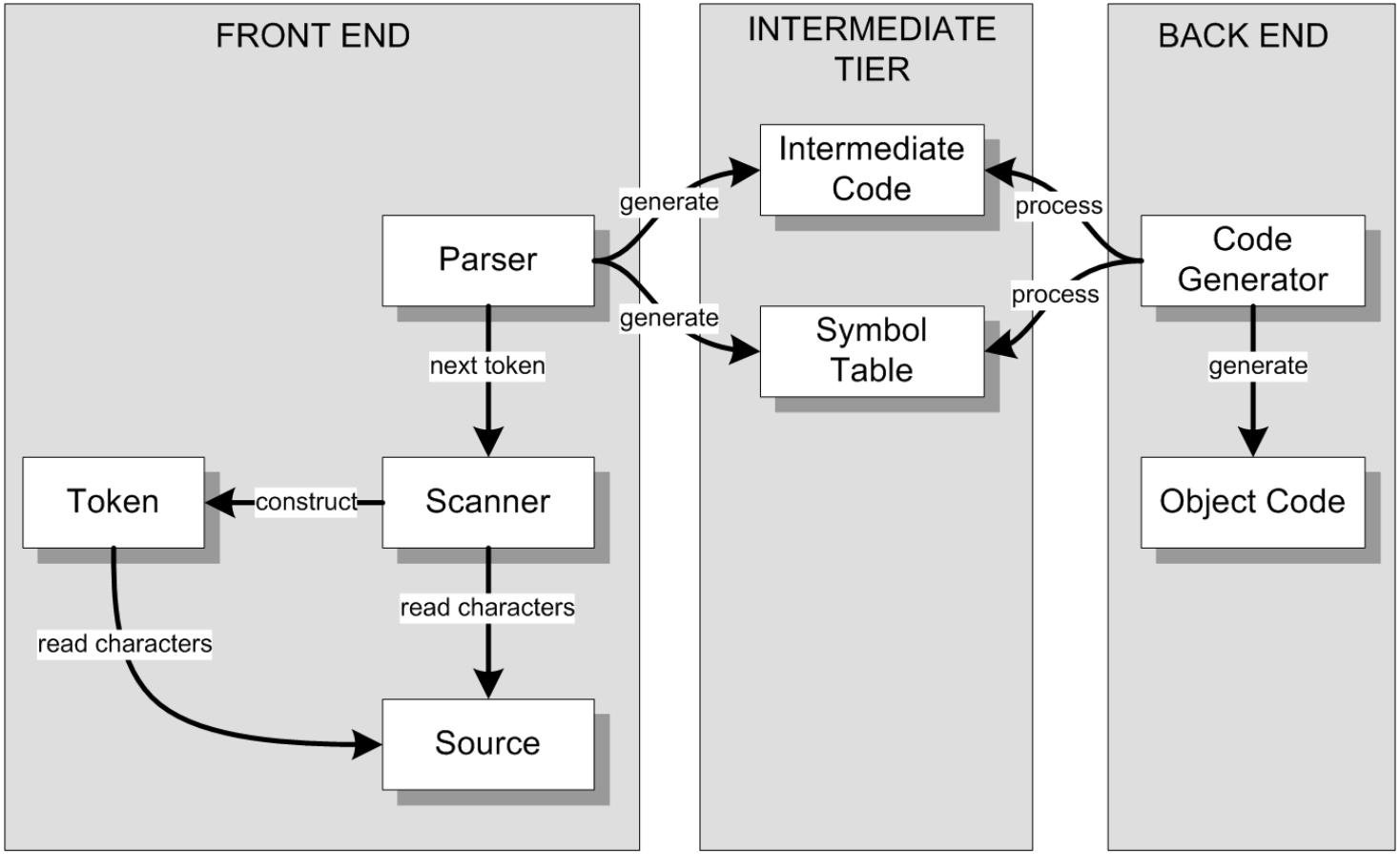
© R. Mak

Scanner

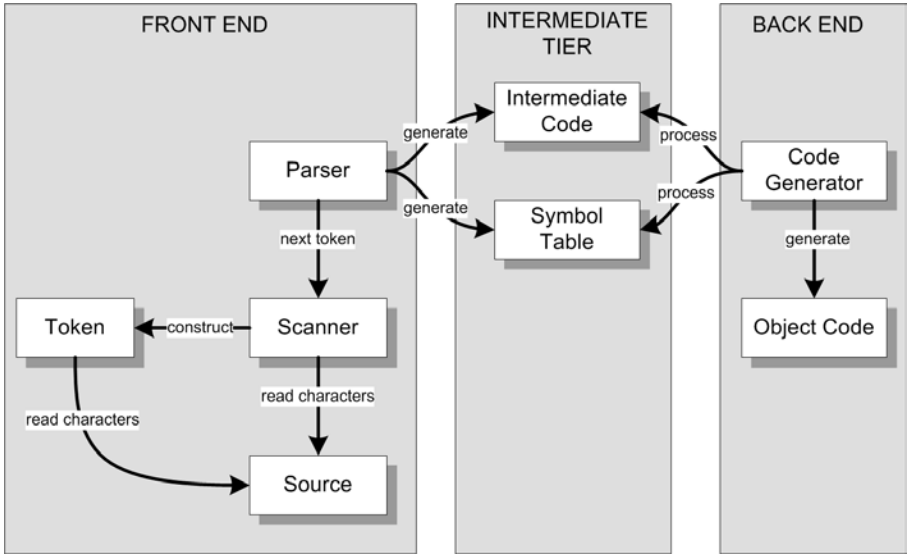
- Reads characters sequentially from the source in order to construct and return the next token whenever requested by the parser.
- Knows the syntax of the source language's tokens.
- What the scanner does is called scanning.
  - It *scans* the source program in order to extract tokens.
  - AKA lexical analyzer

Conceptual Design (Version 2)

- We can architect a compiler with three major parts:



Major Parts of a Compiler



- Front end
  - Parser, Scanner, Source, Token
- Intermediate tier
  - Intermediate code (icode)
    - "Predigested" form of the source code that the back end can process efficiently.
    - Example: parse trees
    - AKA intermediate representation (IR)
  - Symbol table (symtab)
    - Stores information about the symbols (such as the identifiers) contained in the source program.
- Back end
  - Code generator
    - Processes the icode and the symtab in order to generate the object code.

Only the front end needs to be source language-specific.

The intermediate tier and the back end can be language-independent!

What Else Can Compilers Do?

- Compilers allow you to program in a high-level language and think about your algorithms, not about machine architecture.
- Compilers provide language portability.
  - You can run your C++ and Java programs on different machines because their compilers enforce language standards.

What Else Can Compilers Do? *cont'd*

- Compilers can optimize and improve the execution of your programs.
  - Optimize the object code for speed.
  - Optimize the object code for size.
  - Optimize the object code for power consumption.