CSE302: Compiler Design

Instructor: Dr. Liang Cheng

Department of Computer Science and Engineering

P.C. Rossin College of Engineering & Applied Science

Lehigh University

January 16, 2007

Classroom Interactions

- I encourage you to raise questions anytime
- I raise questions
- Major purposes
    - Group-based discussion
    - More efficient in-class learning: learning pattern

Office Hours

- I encourage you to see me if you have any questions
  - Office hours: Fridays from 1 PM to 4 PM (PL326), or by appointment via email
  - chengATcseDOTlehighDOTedu

Today’s Outline

- Course information
- Introduction (Chapter 1)
- Summary and homework

Objectives

- Be able to
  - describe the theory and practice of compilation, in particular
    - Lexical analysis
    - Parsing,
    - Code generation and optimization
  - design a compiler for a concise programming language
- Prerequisites
  - CSE 109: Systems Software
  - CSE318: Automata and Formal Grammars.

Textbook and Languages

- Textbook
  - Compilers: Principles, Techniques, and Tools (2nd Edition) by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Addison Wesley, Boston, MA, 2006. ISBN 0321486811
- Languages
  - C, C++, Java
- Attendance at lecture is required

Grading

- Homework: 20%
  - Due Monday 11:55 PM. No late hand-in homework will be accepted.
  - Submit your work through the Blackboard course website.
- Programming projects: 30%
- Midterm exam: 20%
- Final exam: 30%
  - All exams are open-book ones.

Project Overview

- Individual projects
- Multi-stage compiler design projects
- Academic integrity
  - All graded work should be your own!

Other Homework and Exam Related Issues

- If you'd like to request homework and exam date changes due to some reasons
  - Email me a request at least two weeks ahead of the scheduled deadline
- Accommodations for students with disabilities
  - Contact both me and the Office of Academic Support Services, University Center 212 (610-758-4152)

Course Information

- Course website
  - http://www.cse.lehigh.edu/~cheng
- Course syllabus
  - http://www.cse.lehigh.edu/~cheng/Teaching/CSE302-07/syllabus.html
  - Including the course schedule
    - www.cse.lehigh.edu/~cheng/Teaching/CSE302-07/schedule.html
- For each lecture's slides
  - A preparation version will be uploaded to the course schedule webpage about 10 hours before the lecture
  - A after-class version will be uploaded to the Blackboard System after each lecture
    - Based on materials covered in the class

Outline

- Course information
- Introduction (Chapter 1)
- Summary and homework

Three Questions about Compilers

- What is a compiler?
  - A language processor: source lang -> target lang
- Is it important for people to study compiler design issues?
  - Software running now was compiled by some compilers
- Is it useful for me to learn compiler design techniques?
  - Touch upon programming languages, computer architecture, language theory, algorithms, and software engineering
  - Applicable to many domains

Language Processors

- Language translation
  - Report errors detected
- Compiler vs. interpreter
- Java language processor
  - A hybrid processor
- Language processing systems
  - Preprocessor, compiler, assembler, linker/loader

A Quick Question

- What are the specific things that need to be processed by the language processing system?
    - What is the definition of a language?

Introduction to Language Definition

- Language definition or language specifications
  - What does it looks like?
  - http://java.sun.com/docs/books/jls/
- Who must use language definitions?
  - Language designers
  - Implementors
  - Programmers (the users of the language)

Syntax and Semantics

- Syntax
  - The form or structure of the expressions, statements, and program units
    - **while** (EXPRESSION) { STATEMENTS; }
  - Java syntax: the grammar for Java
    - http://java.sun.com/docs/books/jls/third_edition/html/syntax.html
- Semantics
  - The meaning of the expressions, statements, and program units
    - http://java.sun.com/docs/books/jls/

Terms for Describing Syntax

- A language is a set of sentences
- A sentence is a string of characters, composed of lexemes, over some alphabet
- A lexeme is the lowest level syntactic unit of a language described by a lexical specification
- A token is a category/abstraction of lexemes

Java Lexeme Examples

- Java lexical specification
  - http://java.sun.com/docs/books/jls/third_edition/html/lexical.html
- Java identifier definition
  - "An identifier is an unlimited-length sequence of Java letters and Java digits, the first of which must be a Java letter.
  - An identifier cannot have the same spelling (Unicode character sequence) as a keyword (§3.9), Boolean literal (§3.10.3), or the null literal (§3.10.7)."

Phases of Compilation

- Front end: analysis
  - Scanner
  - Parser
  - Semantic analyzer
  - Intermediate-code generator
- Back end: synthesis
  - Code optimizer (optional)
  - Code generator
- Symbol table

The Scanner

- Also called the Lexer
- How it works:
  - Reads characters from the source program.
  - Groups the characters into **lexemes** (sequences of characters that "go together").
  - Each lexeme corresponds to a **token**;
    - the scanner returns the next token (plus maybe some additional information) to the parser.
  - The scanner may also discover lexical errors (e.g., erroneous characters).

The Parser

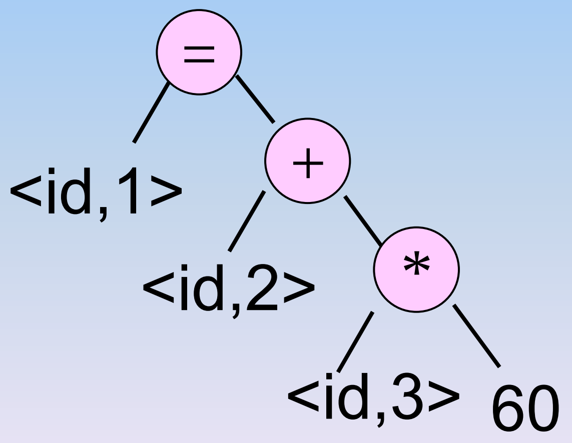
- **Input:** sequence of tokens from lexical analysis
- **Output:** parse tree of the program
  - Parse tree is generated if the input is a legal program
  - If input is an illegal program, syntax errors are issued
  - Instead of parse tree, some parsers produce directly: abstract syntax tree (AST) + symbol table, or intermediate code

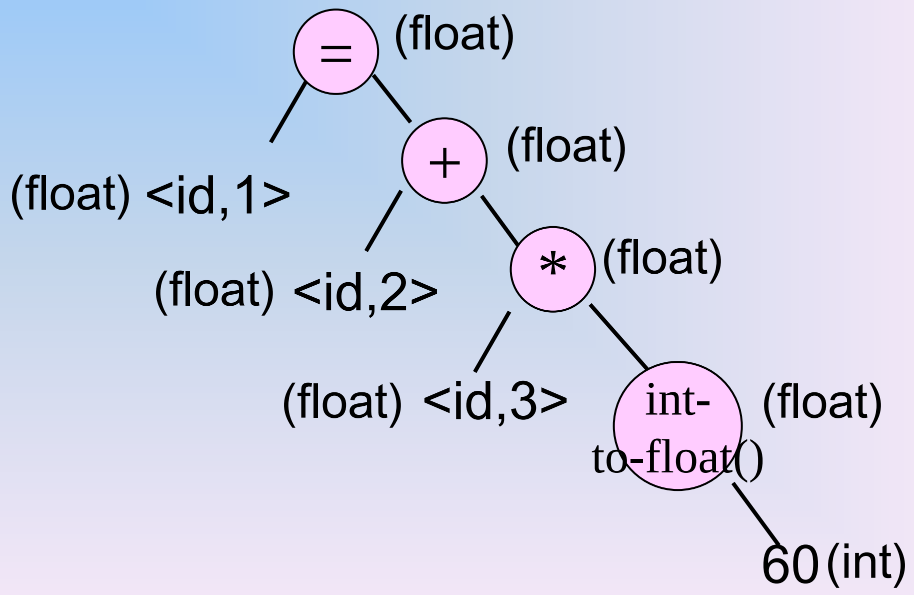
Parser vs. Scanner

Phase	*Input*	*Output*
Scanner	String of characters	String of tokens
Parser	String of tokens	Parse tree, AST, int. code

The Semantic Analyzer

- Checks for "static semantic" errors, e.g., type errors
- Annotates and/or changes the abstract syntax tree based on the attribute grammar
  - Annotate a node that represents an expression with its type.
  - Example with before and after:





Intermediate Code Generator

- Translates from abstract-syntax tree to intermediate code
  - One possibility is 3-address code.
  - Here's an example of 3-address code for the abstract-syntax tree shown previously:

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

The Code Generator

- Generates object code from (optimized) intermediate code

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

Compiler Construction Tools

- Scanner generators
- Parser generators
- Syntax-directed translation engines
- Code-generator generators
- Data-flow analysis engines
- Compiler-construction toolkits

Reading Assignments

- Section 1.3
  - The evolution of programming languages
- Section 1.4
  - The science of building a compiler
- Section 1.5
  - Application of compiler technology
    - Implementation of high-level programming languages
    - Optimization/design for existing/new computer architecture
    - Program translations
    - Building software productivity tools
- Section 1.6
  - Programming language basics
    - Static/dynamic scoping, parameter passing, etc.

Outline

- Course information
- Introduction (Chapter 1)
- Summary and homework