Introduction to Compiler Construction

Robert van Engelen

http://www.cs.fsu.edu/~engelen/courses/COP5621
Syllabus

• Prerequisites: COP4020 or equivalent
• Other material: “The Java™ Virtual Machine Specification” SE 8 and class handouts
• Grade breakdown:
  – Exams (three midterm, one final) (60%)
  – Four project assignments (40%)
  – Homework for extra credit (at most 4%)
Syllabus, Assignments, and Schedule

http://www.cs.fsu.edu/~engelen/courses/COP5621
Objectives

• Be able to build a compiler for a (simplified) (programming) language
• Know how to use compiler construction tools, such as generators of scanners and parsers
• Be familiar with assembly code and virtual machines, such as the JVM, and bytecode
• Be able to define LL(1), LR(1), and LALR(1) grammars
• Be familiar with compiler analysis and optimization techniques
• … learn how to work on a larger software project!
Compilers and Interpreters

• “Compilation”
  – Translation of a program written in a source language into a semantically equivalent program written in a target language
  – Oversimplified view:
Compilers and Interpreters (cont’d)

• “Interpretation”
  – Performing the operations implied by the source program
  – Oversimplified view:

![Diagram showing the interaction between source program, interpreter, input, output, and error messages.]

- Source Program
- Interpreter
- Input
- Output
- Error messages
The Analysis-Synthesis Model of Compilation

- There are two parts to compilation:
  - *Analysis* determines the operations implied by the source program which are recorded in a tree structure
  - *Synthesis* takes the tree structure and translates the operations therein into the target program
Other Tools that Use the Analysis-Synthesis Model

- Editors (syntax highlighting)
- Pretty printers (e.g. Doxygen)
- Static checkers (e.g. Lint and Splint)
- Interpreters
- Text formatters (e.g. TeX and LaTeX)
- Silicon compilers (e.g. VHDL)
- Query interpreters/compilers (Databases)
Preprocessors, Compilers, Assemblers, and Linkers

Skeletal Source Program

Preprocessor

Source Program

Compiler

Target Assembly Program

Assembler

Relocatable Object Code

Linker

Libraries and Relocatable Object Files

Absolute Machine Code

Try for example:
gcc -v myprog.c
## The Phases of a Compiler

<table>
<thead>
<tr>
<th>Phase</th>
<th>Output</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programmer (source code producer)</strong></td>
<td>Source string</td>
<td>A=B+C;</td>
</tr>
<tr>
<td><strong>Parser (performs syntax analysis based on the grammar of the programming language)</strong></td>
<td>Parse tree or abstract syntax tree</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B   C</td>
</tr>
<tr>
<td><strong>Semantic analyzer (type checking, etc)</strong></td>
<td>Annotated parse tree or abstract syntax tree</td>
<td>int2fp B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ t1 C t2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>:= t2 A</td>
</tr>
<tr>
<td><strong>Intermediate code generator</strong></td>
<td>Three-address code, quads, or RTL</td>
<td>int2fp B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ t1 #2.3 A</td>
</tr>
<tr>
<td><strong>Optimizer</strong></td>
<td>Three-address code, quads, or RTL</td>
<td>MOVF #2.3, r1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADDF2 r1, r2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOVF r2, A</td>
</tr>
<tr>
<td><strong>Code generator</strong></td>
<td>Assembly code</td>
<td>ADDF2 #2.3, r2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOVF r2, A</td>
</tr>
<tr>
<td><strong>Peephole optimizer</strong></td>
<td>Assembly code</td>
<td></td>
</tr>
</tbody>
</table>
The Grouping of Phases

• Compiler *front* and *back ends*:
  – Front end: *analysis* (*machine independent*)
  – Back end: *synthesis* (*machine dependent*)

• Compiler *passes*:
  – A collection of phases is done only once (*single pass*) or multiple times (*multi pass*)
    • Single pass: usually requires everything to be defined before being used in source program
    • Multi pass: compiler may have to keep entire program representation in memory
Compiler-Construction Tools

• Software development tools are available to implement one or more compiler phases
  – Scanner generators
  – Parser generators
  – Syntax-directed translation engines
  – Automatic code generators
  – Data-flow engines
Outline

• Introduction
• A simple One-Pass Compiler for the JVM
• Lexical Analysis and Lex/Flex
• Syntax Analysis and Yacc/Bison
• Syntax-Directed Translation
• Static Semantics and Type Checking
• Run-Time Environments
• Intermediate Code Generation
• Target Code Generation
• Code Optimization