Introduction to Compiler Construction

Robert van Engelen

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

Syllabus

- Prerequisites: COP4020
- Four exams (60%) and continuous programming assignments (40%)
- For up-to-date info visit: http://www.cs.fsu.edu/~engelen/courses/COP5621

Assignments and Schedule

http://www.cs.fsu.edu/~engelen/courses/COP5621/assign.html
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 virtual machines, such as the JVM and Java 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

Input
Source Program
Compiler
Error messages
Target Program
Output

Compilers and Interpreters (cont’d)

- "Interpretation" – Performing the operations implied by the source program

Source Program
Input
Interpreter
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

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>Compilation (source code producer)</td>
<td>Source string</td>
<td><code>#include &lt;stdio.h&gt;</code></td>
</tr>
<tr>
<td>Scanner</td>
<td>Token string</td>
<td><code>- #define</code></td>
</tr>
<tr>
<td>Syntax-directed analysis</td>
<td>Parse tree or abstract syntax tree</td>
<td><code>- </code></td>
</tr>
<tr>
<td>Semantic analyzer (type checking, etc)</td>
<td>Annotated parse tree or abstract syntax tree</td>
<td><code>- </code></td>
</tr>
<tr>
<td>Intermediate code generator (three-address code, quads, or RTL)</td>
<td>Three-address code, quads, or RTL</td>
<td><code>- </code></td>
</tr>
<tr>
<td>Optimizer</td>
<td>Three-address code, quads, or RTL</td>
<td><code>- </code></td>
</tr>
<tr>
<td>Code generator</td>
<td>Assembly code</td>
<td><code>ADD r1, r2</code></td>
</tr>
<tr>
<td>Peephole optimizer</td>
<td>Assembly code</td>
<td><code>MOV r3, r2</code></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

• Ch. 1: Introduction
• Ch. 2: A simple One-Pass Compiler for the JVM
• Ch. 3: Lexical Analysis and Lex/Flex
• Ch. 4: Syntax Analysis and Yacc/Bison
• Ch. 5: Syntax-Directed Translation
• Ch. 6: Type Checking
• Ch. 7: Run-Time Environments
• Ch. 8: Intermediate Code Generation
• Ch. 9: Code Generation
• Ch.10: Code Optimization