Introduction to Compiler Construction

Robert van Engelen

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

Syllabus

• Prerequisites: COP4020
• Other material: “The Java™ Virtual Machine Specification”, 2nd edition and class handouts
• Four exams (60%) and continuous programming assignments (40%)
• For more up-to-date info: 
  http://www.cs.fsu.edu/~engelen/courses/COP5621
Objectives

- Know how to build a compiler for a (simplified) (programming) language
- Know how to use compiler construction tools, such as generators for scanners and parsers
- Be familiar with virtual machines, such as the JVM and Java bytecode
- Be able to write LL(1), LR(1), and LALR(1) grammars (for new languages)
- 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

Compilers and Interpreters (cont’d)

• “Interpretation”
  – Performing the operations implied by the source program
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**
- **Compiler**
- **Assembler**
- **Linker**

Source Program

Target Assembly Program

Relocatable Object Code

Absolute Machine Code

**Try for example:**
```
gcc -v myprog.c
```
The Grouping of Phases

• Compiler front and back ends:
  – Analysis (*machine independent* front end)
  – Synthesis (*machine dependent* back end)

• Passes
  – A collection of phases may be repeated 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