CS 5363, Fall 2013
Lecture Notes #1

Reading / Source Material

Scott (3rd ed.): 1.1–1.5
Cooper (2nd ed.): 1

Vocabulary

Be able to define the following terms as well as compare and contrast related terms:
dynamic dispatch, recursion, compiler, interpreter, read-eval-print loop, front-end, back-end, 'compiled' language, 'interpreted' language, virtual machine, just-in-time compiler, source language, source code, target language, bytecode, machine code, optimizer, compiler phase

Objectives

1. recursion, dynamic dispatch, and object-oriented tree data structures
   (a) trace the execution of code using dynamic dispatches.
   (b) trace the execution of code using recursive function calls.
   (c) explain how recursion and dynamic dispatch can be used to print, evaluate, and copy an expression abstract syntax tree.
2. compilers vs. interpreters (practical)
   (a) edit, compile, and run a program from the command-line.
   (b) edit, and run a program with an interpreter from the command-line.
   (c) interact a read-eval-print-loop (REPL).
3. compilers vs. interpreters (conceptual)
   (a) explain the difference between an interpreter and a compiler and the benefits of each
   (b) explain how the standard Java Virtual Machine behaves like an interpreter.
   (c) explain how the standard Java implementation utilizes compilers and interpreters
   (d) explain how many language implementations actually combine aspects of compilation and interpretation
   (e) compare the Java language implementation (source code, the Java source compiler, the Java bytecode representation, the Java Virtual Machine, JIT compiler, interpreter) with the pure interpreter and pure compiler models
   (f) explain why, technically, a language isn't 'compiled' or 'interpreted'

    (g) identify language characteristics typically associated with the language being interpreted

4. Compiler Structure

    (a) be able to identify the parts of a typical 3-phase optimizing compiler and how those relate to the parts that will need to be implemented for the compiler project.

Outline

1. Using Dynamic Dispatch and Recursion
    (a) C++ example of dynamic dispatch
    (b) using g++ from the command-lien
    (c) C++ example of recursion
    (d) scala's read-eval-print loop (REPL)
    (e) scala example of expression AST
    (f) interpreting scala program file
    (g) compiling scala to bytecode
    (h) running bytecode with 'java' and with 'scala'
2. Programming Language Implementation Strategies
    (a) pure interpretation
    (b) pure compilation
    (c) real world:
        i. translator + virtual machine
            A. perl, python, tcl
            B. Java
            C. .Net
    (d) 'interpreted languages' vs. 'compiled languages'
3. Compiler Structure
    (a) front-end
        i. scanner
        ii. parser
        iii. semantic analysis / type checking
    (b) optimizer (optional)
    (c) back-end
        i. instruction selection
        ii. instruction scheduling
        iii. register allocation

1 Examples

See github repository:

web URL `https://github.com/UTSA-CS-5363-Fall2013/lecture-examples`

git URL `https://github.com/UTSA-CS-5363-Fall2013/lecture-examples.git`

Questions

- If I have a C++ source file source.cc and run the command `g++ -o program source.cc` and it compiles successful, what file will be produced and what will its contents be?
- If I have a Java source file source.java containing the class MyClass (belonging to the default package) and run the command `javac source.java` what file will be produce and what will its contents be?
- Trace the execution of the example programs and determine what their output will be without running them in a compiler.
- What are the three main phases of a typical optimizing compiler and what does each do?
- What 3 passes make up the front-end of a compiler and what does each do?
- What 3 passes make up the back-end of a compiler and what does each do?
- If you want to take an existing compiler (say, C to i386) that follows the structure described in the text book and change it to accept a new source language (to make, e.g., a C++ to i386 compiler), which phase of the compiler would you change?
- If you want to take an existing compiler (say, C to i386) that follows the structure described in the text book and change it to target a different architecture (to make, e.g., a C to PowerPC compiler), which phase of the compiler would you change?
- What are the advantages of interpretation over compilation? What are the advantages of compilation over interpretation?
- How does Oracle's standard implementation of Java for x86 work? What three languages does it process and/or produce? How does it use compilers? How does it use an interpreter?
- What do people mean when they say that a language (e.g., python) is an interpreted language?