

CS5810: Compiler Design and Implementation

Overview of the Course

Basic Information

- Course home page:
<http://www.cs.wmich.edu/~zjiang/CS5810/index.html>
 Or follow the link at
<http://www.cs.wmich.edu/~zjiang>
 - Slides, assignments, etc
 - I will not have handouts in class; get them from the web
- Textbook:
 - Dragon book
- Instructor: James Yang, zjiang.yang@wmich.edu
- Class time: 600-830pm, Monday @ C136
- Office hours: 200-300pm, MW @B-257 or by appointment.

Lecture 1, CS581 Spring 2007

2

Project, Homework, Final Exam & Others

- Project-based course:
 - Project: 70%
 - Final: 20%
 - Class Participation 10%
- Project Grading:
 - No syntax error: 50%
 - Open test data: 35%
 - Close test data: 5%
 - Additional requirements: 10%

Lecture 1, CS581 Spring 2007

3

The Course Project

- A big project
- ... in 4 not difficult but also not easy parts
- Start early!

Lecture 1, CS581 Spring 2007

4

Project Cheating Policy & Others

- Your program cannot print out the output directly from an input according to an open test data. (cheating violation)
- Later project/homework Policy: 4 late days policy.
Start a project as early as you can!!! Don't wait until the open test data are available!!!

Lecture 1, CS581 Spring 2007

5

Grading

- Graduate class model
 - No memorizing facts, algorithms, formulas, etc. (but you need to understand)
 - You will have to be independent in this class and write your PAs by yourself
(You can discuss PAs with some other students but cannot sit together to write PAs!!!)

Lecture 1, CS581 Spring 2007

6

Class-taking Technique

- Read the book
 - Not all material will be covered in class
 - The tests will cover both lecture and reading
- Come to class
 - I take test questions from low-attendance classes
 - Pop quizzes
- Do the programming assignments
 - Good practice for the tests
 - 70%!

Lecture 1, CSS81 Spring 2007

7

Compilers

- What is a compiler?
 - A program that translates an executable program in one language into an executable program in another language
 - The compiler should improve the program, in some way
- What is an interpreter?
 - A program that reads an executable program and produces the results of executing that program
- C is typically compiled, Scheme is typically interpreted
- Java is compiled to bytecodes (code for the Java VM)
 - which are then interpreted
 - Or a hybrid strategy is used
 - Just-in-time compilation

Lecture 1, CSS81 Spring 2007

8

Why Study Compilation?

- Compilers are important
 - Responsible for many aspects of system performance
- Compilers are interesting
 - Compilers include many applications of theory to practice
 - Writing a compiler exposes practical algorithmic & engineering issues
- Compilers are everywhere
 - Many practical applications have embedded languages
 - Commands, macros, formatting tags ...
 - Many applications have input formats that look like languages

Intrinsic Merit

- Compiler construction poses challenging and interesting problems:
 - Compilers must do a lot but also run quickly
 - Compilers have primary responsibility for run-time performance
- Computer architects perpetually create new challenges for the compiler by building more complex machines
 - Compilers must hide that complexity from the programmer
- A successful compiler requires mastery of the many complex interactions between its constituent parts

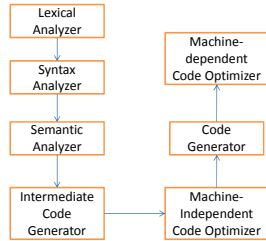
Intrinsic Interest

- Compiler construction involves ideas from many different parts of computer science
- Artificial intelligence
 - Greedy algorithms, Heuristic search techniques
- Algorithms
 - Graph algorithms, union-find Dynamic programming
- Theory
 - DFAs & PDAs, pattern matching, Fixed-point algorithms
- Systems
 - Allocation & naming, Synchronization, locality
- Architecture
 - Pipeline & hierarchy management, Instruction set use

Why Does This Matter Today?

- In the last 2 years, most processors have gone multicore
- The era of clock-speed improvements is drawing to an end
 - Faster clock speeds mean higher power (n^2 effect)
 - Smaller wires mean higher resistance for on-chip wires
 - For the near term, performance improvement will come from placing multiple copies of the processor (core) on a single die
 - Classic programs, written in old languages, are not well suited to capitalize on this kind of multiprocessor parallelism
 - Parallel languages, some kinds of OO systems, functional languages
 - Parallel programs require sophisticated compilers

The Structure of a Compiler



Lexical Analysis (Scanning)

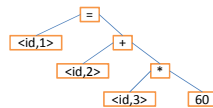
`position = initial + rate * 60`

- Reads the stream of characters and groups the characters into meaningful sequences called lexemes
 - Produces tokens of the form
 - <token-name, attribute-value>
 - <id,1> <=> <id,2> <+> <id,3> <*> <60>

Syntax Analysis (Parsing)

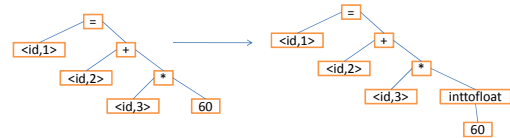
`<id,1> <=> <id,2> <+> <id,3> <*> <60>`

- Create a tree-like representation that depicts the grammatical structure of the token stream
- Syntax tree
 - Interior node represents an operation
 - Children represent the arguments



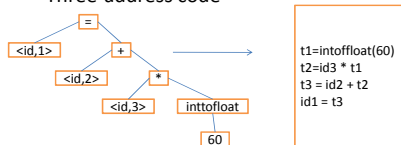
Semantic Analysis

- Use syntax tree and symbol table to check the source program for semantic consistency with the language definition
 - Type checking



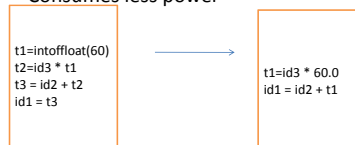
Intermediate Code Generation

- Many compiler generate an explicit machine-like intermediate representation
 - Easy to produce
 - Easy to translate into the target machine
 - Three-address code



Code Optimization

- Improve the intermediate code so that better target code will result
 - Faster
 - Shorter
 - Consumes less power



Code Generation

- Maps an intermediate representation into the target language

