Topic #11:
Compilers

CSE 413, Autumn 2004
Programming Languages

http://www.cs.washington.edu/education/courses/413/04au/

Credits

• Much of the material in the following lectures is
  » derived from Doug Johnson, CSE 413…
  » derived from Hal Perkins, CSE 413 and CSE 582…
  » derived from…
  • Cornell CS 412-3 (Teitelbaum, Perkins)
  • Rice CS 412 (Cooper, Kennedy, Torczon)
  • UW CSE 401 (Chambers, Ruzzo, et al)

References

• Primary Reference
  » Sebesta text, Chapters 3 and 4
  » READ 3-3.3, SKIM 3.4-3.5
  » READ Chapter 4

• Other references
  » Engineering a Compiler by Keith Cooper & Linda Torczon
  » Modern Compiler Implementation in Java, by Appel

Why are we doing this?

• Execute this ...
  ```java
  int nPos = 0;
  int k = 0;
  while (k < length) {
    if (a[k] > 0) {
      nPos++;
    }
  }
  ```

Interpreters & Compilers

• Interpreter

• Compiler

Common Issues

• Compilers and interpreters both must read the input – a stream of characters – and “understand” it; analysis

  ```java
  while (k < length) {
    if (a[k] > 0) {
      nPos++;
    }
  }
  ```
### Interpreter
- Interpreter
  - Execution engine
  - Program execution interleaved with analysis
    ```java
    while (running) {
      analyze_next_statement;
      execute_that_statement;
    }
    ```

### Compiler
- Read and analyze entire program
- Translate to semantically equivalent program in another language
  - Presumably easier to execute or more efficient
  - Should “improve” the program in some fashion
- Offline process

### Typical Implementations
- Compilers
  - FORTRAN, C, C++, Java, C#, COBOL, etc. etc.
- Interpreters
  - PERL, Python, awk, sed, sh, csh, postscript printer, Java VM
  - Functional languages like Scheme and Smalltalk where the environment is dynamic

### Hybrid approaches
- Well-known example: Java
  - Compile Java source to byte codes – Java Virtual Machine language (.class files)
  - Execution

### Why Study Compilers? Programmer
- Become a better programmer
  - Insight into interaction between languages, compilers, and hardware
  - Understanding of implementation techniques
  - What is all that stuff in the debugger anyway?
  - Better intuition about what your code does
- You might even write a compiler some day!

### Why Study Compilers? Designer
- Compiler techniques are everywhere
  - Parsing (little languages, interpreters)
  - Database engines
  - AI: domain-specific languages
  - Text processing
    - Tex/LaTeX, dvi, Postscript, pdf
  - Hardware: VHDL, model-checking tools
Why Study Compilers? *Theoretician*

- Fascinating blend of theory and engineering
  - Direct applications of theory to practice
  - Some very difficult problems (NP-hard or worse)

Why Study Compilers? *Education*

- Ideas from many parts of CSE
  - AI: Greedy algorithms, heuristic search
  - Algorithms: graph algorithms, dynamic programming, approximation algorithms
  - Theory: Grammars DFAs and PDAs, pattern matching, fixed-point algorithms
  - Systems: Allocation & naming, synchronization, locality
  - Architecture: pipelines & hierarchy management, instruction set use
- Application to many other problem domains

Structure of a Compiler

- First approximation
  - Front end:
    - Read source program and understand structure/meaning
  - Back end:
    - Generate equivalent target language program

Intermediate Representation (IR)

- Front end:
  - Source → Front End → Back End → Target

Implications

- Front End
  - Split into two parts
    - Both can be generated automatically or by hand
      - Source language specified by a formal grammar
      - Tools read the grammar and generate scanner & parser (either table-driven or hard coded)
Tokens

- Token stream: Each significant lexical chunk of the program is represented by a token
  - Operators & Punctuation: {}, [], !, +, -, *, :, ...
  - Keywords: if, while, return, goto
  - Identifiers: (variables, procedure names…)
  - Constants: (int, floating-point character, string, …)

Scanner Example

- Input text
  ```
  // this line is a simple comment
  if (x > y) y = 42;
  ```
- Token Stream
  ```
  IF [LPAREN ID(x) OP_GE ID(y) RPAREN ID(y) OP_ASSIGN INT(42) SCOLON]
  ```

Parser Output (IR)

- Many different forms
- Common output from a parser is an abstract syntax tree
  - Essential meaning of the program without the syntactic noise

Parser Example

- Token Stream Input
  ```
  IF [LPAREN ID(x) GEQ ID(y) RPAREN ID(y) BECOMES INT(42) SCOLON]
  ```
- Abstract Syntax Tree
  ```
  ifStmt >= ID(x) ID(y) assign ID(y) INT(42) SCOLON
  ```

Static Semantic Analysis

- During or (more common) after parsing
  - Type checking
  - Check for language requirements
  - Preliminary resource allocation
  - Collect other information needed by back end analysis and code generation

Back End

- Responsibilities
  - Translate IR into target machine code
  - Should produce fast, compact code
  - Should use machine resources effectively
    - Registers
    - Instructions
    - Memory hierarchy
Back End Structure

- Typically split into two major parts with subphases
  - “Optimization” – code improvements
  - Code generation

The Result

```
if (x >= y)
y = 42;
```

```
mov eax, [ebp+16]
cmp eax, [ebp-8]
jl L17
mov [ebp-8], 42
L17:
```

Some Ancient History

- 1950’s. Existence proof
  - FORTRAN I (1954) – competitive with hand-optimized code
- 1960’s
  - New languages: ALGOL, LISP, COBOL
  - Formal notations for syntax
  - Fundamental implementation techniques
    - Stack frames, recursive procedures, etc.

Some Later History

- 1970’s
  - Syntax: formal methods for producing compiler front-ends; many theorems
- 1980’s
  - New languages (functional; Smalltalk & object-oriented)
  - New architectures (RISC machines, parallel machines, memory hierarchy issues)
  - More attention to back-end issues

Some Recent History

- 1990’s – now
  - Compilation techniques appearing in many new places
    - Just-in-time compilers (JITs)
    - Whole program analysis
  - Phased compilation – blurring the lines between “compile time” and “runtime”
  - Compiler technology critical to effective use of new hardware (RISC, Itanium, complex memories)