6.035
Project 1:
Scanner/Parser

Jason Ansel

Spring 2010 – Massachusetts Institute of Technology

Your Background

- What are your past experiences with...
  - Regular expressions?
  - Context free grammars?
  - Lexers/Parsers?
  - Compilers?

My Responsibilities

- Help you
- Grading (hopefully largely automated)
  - I will try to release my grading scripts
  - Make sure your code works with them
- Office hours?
- Group meetings? (For projects 2-5)

Grading

- 60% Projects
  - **5% P1 (you are here)**
  - 7.5% P2
  - 10% P3
  - 7.5% P4
  - 30% P5
- 30% Quizzes
- 10% Mini-Quizzes (each lecture, 2 so far)

Grading (Project 1)

- 75% Automated testing
  - About half of the test cases provided to you
    - 25% of grade
  - Other half hidden
    - 50% of grade
- 25% Write-up / Documentation / Code Design
  - More attention given when automated tests fail
  - 2-4 pages for first project

Why 6.035

- Many disciplines are employed in a compiler
- Bridge abstraction layers
  - Between high-level language and architecture
  - Become more efficient programmers
- Learn to design and use some useful tools
  - Language recognition
  - Tree manipulation
  - Pattern recognition
  - Optimization and parallelization frameworks
- Build a large project in a team (Proj 2-5)

Project

- Design a complete optimizing compiler for our Decaf Language targeting x86-64.
- Open-ended
  - Except for first phase you are not going to be given much.
  - The design process is a very important aspect.
  - Bad designs in early projects will come back to hurt you later.
- Compiler competition at the end of the semester.

Decaf Language

- Simple Imperative Programming Language
  - Array, expressions, methods, control flow
  - No: pointers, classes, floating point
- Sort of like simplified fortran/pascal
- Easier to optimize than more complex language

Lexical Analysis (Scanning)

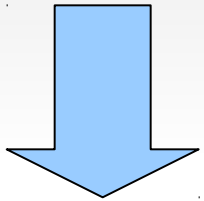
- Covert stream of input characters into tokens.
  - Each token is created without memory of previous tokens
- A token is treated as a unit by later passes.
- The scanner will:
  - Discard whitespace (not in a string or char literal)
  - Denote keywords, integer literals, string and char literals (using delimiters), operators, and identifiers.
  - Report sensible errors for lexically malformed programs. (ANTLR errors mostly OK)

- Traditional tools: lex (unix), flex (gnu rewrite/expansion)

Lexical Analysis

- Example:

```
class Program { void main () {} }
```



```
TK_class ID("Program") LCURLY
  TK_void ID("main") LPAREN RPAREN
  LCURLY RCURLY RCURLY
```

- Don't generate the scanner by hand, use a scanner generator!

Parsing

- Convert stream of tokens into syntax tree
- Expressed as context free grammar
- Converted (by ANTLR) into state machine with stack
- Uses fixed lookahead ('k' in ANTLR)

- Traditional tools: yacc (unix), bison (gnu rewrite/expansion)

Provided Code / Tools

- Optional...
  - but use **some** lexer/paser generator
- Java / Ant / Eclipse
- ANTLR (http://www.antlr.org/)
  - Plenty of documentation online
- See me if you get stuck!

Eliminating Conflicts

- Intuition: The parser does not know what to do given the tokens already seen and the next tokens (lookahead).
- Increasing k can fix some problems (use with caution: k=1, 2, or 3 is sufficient, much higher than that may indicate bad grammar)
- To investigate the source of the conflict you should look at the parser states
- To enable output of parse states in ANTLR, see the ant build file.

Semantic Actions

- Code to execute for a rule.
- Executed after the preceding terminal / non-terminal in the rule is recognized.
- A value can be passed “up” to the enclosing rule.
- For terminals: Value the scanner associated with the terminal is accessible.

```
program : TK_class name:ID
  {System.out.println ("got id: " +
  name.getText()); } LCURLY RCURLY;
```

Shift/Reduce Conflicts

Consider this grammar:

expr: ....

stmt: if_stmt | ...

if_stmt: IF expr THEN stmt

| IF expr THEN stmt ELSE stmt

What is the conflict/ambiguity?

Shift/Reduce Conflicts

- if(x) if(y) win(); else lose();
- Either:
  - if(x){ if(y) win(); else lose(); } (shift)
  - if(x){ if(y) win(); } else lose(); (reduce)
- Most parsers generators default to shift
  - Have directives to change this behavior
  - Throw noisy warnings...

Reduce/Reduce Conflicts

- Consider this grammar

list: /*empty*/

| maybeword

| list word

maybeword: /*empty*/

| word

What is the conflict/ambiguity?

Reduce/Reduce Conflicts

- list → /*empty*/
- list → maybeword → /*empty*/

- Try to create a grammar without conflicts
- Conflict makes life harder

Other Questions?

6.035 Computer Language Engineering

Spring 2010