Structure of the MiniJava Compiler

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>character stream</td>
<td>IR code generation</td>
</tr>
<tr>
<td>PA3</td>
<td>Assem (MIPS)</td>
</tr>
<tr>
<td>lexical analysis</td>
<td>optimization</td>
</tr>
<tr>
<td>PA4</td>
<td>Assem (MIPS)</td>
</tr>
<tr>
<td>tokens “words”</td>
<td>code gen</td>
</tr>
<tr>
<td>PA5</td>
<td>MIPS</td>
</tr>
<tr>
<td>syntactic analysis</td>
<td></td>
</tr>
<tr>
<td>PA6</td>
<td></td>
</tr>
<tr>
<td>AST “sentences”</td>
<td></td>
</tr>
<tr>
<td>semantic analysis</td>
<td></td>
</tr>
<tr>
<td>AST and symbol table</td>
<td></td>
</tr>
</tbody>
</table>

Plan for Today

Studying for the midterm
- review all slides and notes taken in class
- do suggested exercises
- redo any examples we did in class
- reread assigned reading
- the midterm WILL NOT have anything about MIPS

Lexical Analysis, or scanning

Terminology
- regular expressions
- tokens
- DFA
- NFA
- longest match and priority

Techniques
- creating a scanner for a set of tokens

Syntactic Analysis, or parsing

Terminology
- context-free grammars, terminal, non-terminal, symbol, derivation
- syntax-directed translation, actions, attributes
- LR(0) versus LR(1) grammars
- parse trees versus abstract syntax trees
- ambiguity
- LL(k) and left recursion
- top-down and bottom-up parsing
- precedence and associativity
- pre- and post-order depth-first traversals
- error handling
Syntactic Analysis, or parsing cont...

Techniques
- rewrite a grammar so that it is LL(1)
- calculate FIRST and FOLLOW sets
- write a predictive parser with panic mode error handling
- disambiguate expression and list grammars
- create an LR(0) or LR(1) parse table
- parse a string of tokens with an LR parse table

Grammar Hierarchy

Predictive parser for Float Assignment Grammar

```c
void S() { switch (lookahead) {
    case ID: 
        case EOF:// the 2 characters in the FIRST(StmList EOF)
        try { StmList(); match(EOF); } catch { panic(S); } break;
        default: panic(S); break;
    }
}
void StmList() { switch (lookahead) {
    case ID: // FIRST(Stm StmList ) = { ID }
        try { Stm(); StmList(); } catch { panic(StmList); } break;
        case EOF: // FOLLOW(StmList) = { EOF }
        break;
    default: panic(StmList); break;
}
void Stm() { switch (lookahead) {
    case ID: try { match(ID); match(ASSIGN); match(FLOAT);
        } catch { panic(Stm); } break;
    default: panic(Stm); break;
}
```

Example LR Parse Table

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>1</td>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>3</td>
<td>s2</td>
<td>s4</td>
</tr>
<tr>
<td>4</td>
<td>r1</td>
<td>r1</td>
</tr>
</tbody>
</table>