Plan for Today

FIRST, FOLLOW, and predictive parser table recap

Error recovery in predictive parsers

Predictive parsing as a specific subclass of recursive descent parsing
- necessary to remove left-recursion
- might have to left-factor
- complexity comparisons with general parsing

MiniJava compiler
- source language: MiniJava
- target language: MIPS (Thursday)

Constructing the Predictive Parser Table

Algorithm
for each X -> gamma
    for each T in FIRST(gamma)
        table[X,T] = X->gamma
    if gamma is nullable
        for each T in FOLLOW(X)
            table[X,T] = X->gamma

(1) svg -> SVG_START elem_list SVG_END EOF
(2a & b) elem_list -> elem elem_list | epsilon
(3) elem -> RECT_START ... ELEM_END
(4) elem -> CIRCLE_START ... ELEM_END
(5) elem -> LINE_START ... ELEM_END
Error Recovery

Goals
– Provide program with a list of as many errors as possible
– Provide USEFUL error messages
  – appropriate line and position information
  – guidance for fixing the error
– Avoid infinite loops or recursion
– Add minimal overhead to the processing of correct programs

Approaches
– Stop after first error
– Panic mode
– Phrase-level recovery

Predictive parser w/error recovery for float assign 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;
} }
```