**Plan for Today**

- Error recovery for 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
- MeggyJava language intro, especially terminals for PA2
- Working with a partner

---

**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
  - very simple, but unfriendly
- Panic mode
  - skip tokens until a “synchronizing” token is encountered

---

**Panic mode error recovery**

The function for nonterminal X has one clause for each possible production rule for X. A clause includes a case for every character in the FIRST set for the rhs of the production, each character in the FOLLOW set if the rhs is nullable, and calls to match tokens and other nonterminals to process the rhs of the production.

For panic mode, skip tokens until a follow of the nonterminal encountered

```java
// panic method for nonterminal N
def panic_N()
{
    print error;
    while ( scan() not in (FOLLOW(N) union {EOF}) ) {

    }
}
```

---

**Example: simple assignment grammar**

```
S → StmtList EOF
Stm → id ASSIGN float
StmtList → Stm StmtList | ε
```

What is nullable, FIRST, FOLLOW for each nonterminal?
Predictive parser with panic mode error recovery

```c
// Float assignment grammar.
void S() { switch (m_lookahead) {
    case ID: // the 2 characters in the FIRST(StmList EOF)
        try { StmList(); match(EOF); } catch { panic_S(); } break;
    default: panic_S(); break;
}

void StmList() { switch (m_lookahead) {
    case ID: // FIRST( Stm StmList ) = { ID }
        Stm(); StmList(); break;
    case EOF: // FOLLOW(StmList) = { EOF }
        break;
    default: panic_StmList(); break;
}

void Stm() { switch (m_lookahead) {
    case ID: try { match(ID); match(ASSIGN); match(FLOAT); }
            catch { panic_Stm(); } break;
    default: panic_Stm(); break;
}
```

Predictive Parsing Complexity

**LL(k) grammar classes**
- Left-to-right scan
- Left-most derivation
- k tokens of lookahead

**Comparing complexity**
- $O(N^3)$ for general context free grammars, where $N$ is the number of tokens in the stream (Earley parsing algorithm)
- $O(N)$ for predictive parsing

**Requirements for LL(1), for all productions of nonterminal A**
- None of the FIRST(rhs) for A production rules can overlap
- If nullable(A) then FOLLOW(A) must not overlap with FIRST(rhs) for any A-> rhs

Working with a partner

**Why?**
- No programmer is an island.
- The MeggyJava compiler is a significant course project.
- Working with a partner SHOULD encourage planning ahead.
- Two heads are better than one.

**Each of you is ultimately responsible for making the compiler work**
- Come see me if you are having problems.