Today

- Quiz 3
- Exam 1 Overview
- AST (cont)

Assignments

- HW3 due
- Exam 1 next Tues
- HW4 out next Thurs
Exam 1 Overview

Basics:

- Closed book, notes, etc.
- 10% of final grade
- 4 multipart questions

Topics: (Everything in lectures, homework, & readings is fair game)

- Turing machines (basic idea)
- Compilation vs Interpretation
- Lexical Analysis (purpose, input/output, examples)
- Grammars (create, derivations, properties, LL(k))
- Syntax analysis (recursive descent parsing, examples)
- Basic terminology from readings (mainly HW1)
- Likely ask you to write some code
AST: Running Example

\[
\langle \text{stmt\_list} \rangle \ ::= \ \text{VAR ASSIGN} \ \langle \text{expr} \rangle \ \langle \text{stmt\_list\_tail} \rangle \\
\langle \text{stmt\_list\_tail} \rangle \ ::= \ \text{SEMICOLON} \ \langle \text{stmt\_list} \rangle \ | \ \epsilon \\
\langle \text{expr} \rangle \ ::= \ \text{VAR} \ \langle \text{expr\_tail} \rangle \\
\langle \text{expr\_tail} \rangle \ ::= \ \text{PLUS} \ \text{VAR} \ | \ \text{MINUS} \ \text{VAR} \ | \ \epsilon
\]

Parser class with basic methods and member variables:

```java
public class Parser {
    private Lexer lexer;
    private Token currToken;

    public Parser(Lexer lexer) {...}
    // start recursive descent
    public void parse() throws MyPLEXception {...}

    private void advance() throws MyPLEXception {...}
    private void eat(TokenType t, String errmsg) ...
    private void error(err_msg) ...

    private void stmtList() ...
    private void stmtListTail() ...
    private void expr() ...
    private void exprTail() ...
}
```

We’ll be modifying the above functions to build up the AST ...

- include the signatures as needed
In our example, AST might contain nodes (objects) representing:

- statement lists
- an assignment with an identifier and an expression
- an expression with a single variable
- an expression with two variables and an operator

Note we’d also have Expr as a superclass of VarExpr and OpExpr

```java
public class StmtList {
    public ArrayList<AssignStmt> stmts = new ArrayList<>();
}

public class AssignStmt {
    public Token lhs = null; // lvalue (VAR token)
    public Expr rhs = null;  // right-hand side expression
}

public class Expr { // or as an interface
}

public class VarExpr extends Expr {
    public Token var = null; // VAR token
}

public class OpExpr extends Expr {
    public Token leftOperand = null; // VAR token
    public Token operator = null; // PLUS or MINUS token
    public Token rightOperand = null; // VAR token
}
Adding AST creation to our Recursive Descent Parser

```java
public StmtList parse() throws MyPLException {
    StmtList stmtListNode = new StmtList(); // create StmtList node
    advance(); // init lexer
    stmtList(stmtListNode); // descend into stmt_list
    eat(TokenType.EOS, "..."); // ensure EOS
    return stmtListNode; // return AST root node
}

public stmtList(StmtList stmtListNode) throws MyPLException {
    AssignStmt assignNode = new AssignStmt(); // create Assign node
    assignNode.lhs = currToken; // set lhs
    eat(TokenType.VAR, "..."); // ensure VAR
    eat(TokenType.ASSIGN, "..."); // ensure ASSIGN
    assignNode.rhs = expr(); // descend and return rhs
    stmtListNode.smts.add(assignNode); // add Assign node
    stmtListTail(stmtListNode); // descend to tail
}
```

Exercise: Rewrite the remaining recursive descent functions to build the AST

Exercise: Draw the AST (object graph) resulting from the string “A=B+C; B=A”