1 Summary: Compilation

- Structure of the Jex - interpreter.
- Differences if we really have a compiler.
- Other topics in compilation

2 Scanner

- Scanner.java generated from jex.lex.
- Input: Character sequence (The input file).
- Output: Token sequence.
- Checks: Is the input a valid sequence of tokens.
- Code example:
  ({DIGIT})+ { **return** mkToken(NUMLIT, **new** Integer(yytext())); }
- ({DIGIT})+ says:
  A non-empty sequence of digits is a valid token.
- { **return** mkToken(NUMLIT, **new** Integer(yytext())); } says:
  Construct a new token for the digit sequence.
- Important function: nextToken().
- Theory: Regular expressions, Finite state automaton.

3 Parser

- Parser.java generated from jex.cup.
- Input: A token sequence.
- Output: An abstract syntax tree.
- Checks: Is the token sequence valid for the context free gramar.
- Code example:

```
FunDef ::= Type:tp IDENT:id LPAREN RPAREN Statement:s
           {: RESULT = new FunDef(idleft, tp, id, new Tree[0], s); :}
```

- Type:tp IDENT:id LPAREN RPAREN Statement:s says:
  This is a valid function definition.
- {: RESULT = **new** FunDef(idleft, tp, id, **new** Tree[0], s); :} says:
  This is the way to construct the node for the abstract syntax tree.
- Important function: parse() which returns the abstract syntax tree.
- Theory: Grammars, Stack automaton.

4 PrettyPrinter

- Printer.java.
- Input: The abstract syntax tree.
- Output: A nice version of the tree on standard output.
- Code example:

```
public void caseAssign(Tree.Assign tree) {
        print(tree.left);
        System.out.print(" = ");
        print(tree.right);
        System.out.print(";");
}
```

- Important function: prettyPrint(Tree tree).
- Theory: Visitor

5 Analyzer

- Analyzer.java
- Input: The abstract syntax tree.
- Output: The abstract syntax tree, some more fields computed (sym).
- Computes: All the symbols, some run-time information.
- Checks: Each used identifier is defined properly.
- Checks: Some validity constraints (e.g. The type in a formal may not be a variable)
- Important function: analyzeTree(Tree tree).

```
public void caseFunCall(Tree.FunCall tree) {
    for (int i = 0; i < tree.args.length; i ++)
        analyze(tree.args[i], scope, toplevel);
    tree.sym = (JexSymbol) scope.lookup(tree.name);
    if (tree.sym == null)
        Report.error(tree.pos, "function " + tree.name + " undefined");
}
```


6 Interpreter

- Interpreter.java
- Input: The abstract syntax tree.
- Interprets (runs) the program.
- Checks: Types match for functions, methods, operations, variables.
- Important function: interpretTree(Tree tree).

```
public void caseWhile(Tree.While tree) {
    while (!isReturn) {
        JexValue b = interpret(tree.expr);
        if (!b.isBoolean())
            throw new JexException(tree.pos,
                "condition in while not boolean");
        if (!b.getBoolean())
            break;
        val = interpret(tree.body);
    }
}
```


7 Abstract Syntax Tree

- Tree.java
- Classes for every kind of node in the syntax tree

```
public static class Assign extends Tree {
    Tree left;
    Tree right;

    public Assign(int pos, Tree left, Tree right) {
        super(pos);
        this.left = left;
        this.right = right;
    }

    public void apply(Visitor v) {
        v.caseAssign(this);
    }
}
```

- Important function: apply(Visitor v) for use by visitors.
- Theory: Visitors

8 Symbol Table

- Scope.java
- Symbol.java, JexSymbol.java,GlobalSym.java, LocalSym.java,ClassSym.java,FunSym.java.
- The symbol table knows for each identifier the relevant information.
- Important functions: enter(Symbol s), lookup(String name).

9 Run-time Values

- Environment.java
- JexValue, JexBasic, JexObject, JexClass.
- Contains relevant information for run-time.

10 Other Topics in Compilation

- Code Generation.
  - How to generate machine code?
  - How to do environments?
  - Register allocation.
- Optimization
  - Constant folding $\Rightarrow$ Dataflow analysis.
  - Loop invariants $\Rightarrow$ Loop optimizations.
- Object oriented languages.
  - How to implement objects?
  - How to translate dynamic binding?
- Run-time
  - Threads, how to do synchronization fast?
  - Memory management, garbage collection.