

## CS 671 – Test 1 – The Front End

### General Compilers

- Front-end vs. Back-end
- Compiler phases and interfaces
- Compilers vs. Interpreters vs. JITs

### Lexical Analysis

- Tokenizing
- Regular Expressions
- Deterministic Finite Automata
  - Transition Tables
- Non-deterministic finite automata
  - Epsilon closure
- RegExp → NFA → DFA
- Ad-hoc lexers

### Syntactic Analysis

- Context-free grammars
- Derivations: leftmost vs. rightmost
- Ambiguous grammars
  - Eliminating ambiguity
- Parse Trees
- Abstract Syntax Trees
- Top-Down Parsing
- Bottom-Up Parsing
- Recursive descent
- Left recursion vs. right recursion
- Predictive parsing tables
- First, follow sets
- Shift-reduce parsing
- Yacc/Bison
  - Specification file
  - Conflicts: shift/reduce vs. reduce/reduce
  - Precedence rules
- Error recovery
  - Local vs. Global

### Semantic Analysis

- Why we separate syntactic/semantic analysis
- Static vs. dynamic checks
- Typical semantic errors
- Scope (static and dynamic)
- Overloading
- Symbol tables
  - Implementation options
- Bindings
- Types
  - Statically typed
  - Dynamically typed
  - Untyped
  - Benefits, trade-offs of each

- Type checking vs. type inference
- Built-in types
- Type constructors
- Type equivalence – structural vs. name
- Type coercion

### **Activation Records / Stack Frames**

- Invocations vs. instantiations
- Stack pointer vs. frame pointer
- Address space: stack, heap, static data, code
  - Stack allocation vs. heap allocation vs. static allocation
- Handling local variables
- Handling nested procedures
- Passing parameters
  - why we need calling conventions
  - register windows
- Calling sequence
- Caller save vs. callee save
  - Optimizations, leaf procedures
- Passing return values
- Return address – why we need it, when we store it
- Static link vs. dynamic link – why we need each
- Registers vs. memory – tradeoffs, why compilers care

### **Intermediate Representations**

- Reasons for an IR
- What makes a good IR
- Three-address code
  - Shorthand: triples, quadruples
  - Converting sample code into three-address code
- The IR machine
- Multiple IR stages (HIR, MIR, LIR) – features, motivation, examples
- Graphical vs. linear IR
  - DAGs
  - Dependence graphs
  - CFGs
- Stack machine code

### **Basic Blocks and Traces**

- Partitioning code into basic blocks
- Ordering basic blocks
- Trace creation – heuristics, benefits

### **IR CodeGen**

- Converting HIR to LIR
- generate functions (for statements, conditionals, loops, functions, etc)
- Short circuiting