Today

- Quiz 4
- Static Analysis Intro

Announcements

- HW-4 out
- EX-4 out
- Exam-1 Tuesday
Exam 1 Overview

Basics:
- Closed notes, etc.
- Worth 50 points (plus a small amount of extra credit)
- Four multipart questions

Potential Topics: Note covers what we've discussed to this point
- MyPL syntax
- Compilation/interpretation phases, differences
- Lexical analysis: tokens, token streams, lexer
- Context free grammars, derivations
- LL(k) restrictions, common problems to watch out for, rewritings
- Recursive descent parsing
- Abstract syntax trees, generating ASTs within parser
- Associativity and precedence issues
- Role of static analysis ideas (from today)
**Intro to Static Analysis – Important Terminology**

### Denotable Objects
- Items that can be "**named**" in a programming language
- By the programmer (e.g., variables, functions, classes)
- By the language itself (e.g., primitive types, built-in functions)

### Blocks
- A block is a **textual region** of a program (e.g., function body, loop body)
- A block uses syntax to define start and end
- Declarations (e.g., of user-defined denotable objects) occur within “blocks”

### Static vs Dynamic
- Static generally implies at **compile time** (i.e., before runtime)
- Dynamic generally implies at **runtime**

### Scope Rules (aka “visibility” rules)
- Define what declarations are visible in which blocks
- An object is **local** to the block it is declared in
- In general, an object is visible in its local and nested blocks
- To find the declaration, look in the current block and the containing blocks
Static Scope  (aka “lexical” scope)
  • The visibility of objects (names) determined at compile time
  • Based on the text of the source code
  • What we normally think of as scope (visibility)

Dynamic Scope
  • The visibility of objects (names) determined at runtime
  • Based on last association created for the object
  • See textbook and exercises for additional info ...

Most (modern) PLs primarily have static scoping rules
  • some tricky cases though ...
  • e.g., with nested functions, passing code blocks to functions, closures
The goal of static analysis is to:

- Detect errors due to type issues, e.g.:

  ```
  x = 0 + "1"  # int + string not allowed
  if 42 <= true {
      x = 1
  }
  ```

- Detect “use before def” errors, e.g.:

  ```
  var x = 42 + y  # y not defined
  if x > 42 {
      var y = x + 1
  }
  else {
      x = y  # y not defined in this block
  }
  ```

- Detect function call errors, e.g.:

  ```
  fun int add(int x, int y) {
      return x + y
  }
  fun void main() {
      var r1 = add(1, 2, 3)  # wrong number of args
      var r2 = add(3.14, 1)  # wrong argument types
  }
  ```

- and so on ...
Examples of other kinds of errors searched for during static analysis

- duplicate function names
- duplicate user-defined type names
- parameters and fields are unique
- variable shadowing
- main function defined
- etc.

Type errors often based on a set of typing rules

- the rules define how types can be “inferred” (inference rules)
- makes checking expressions and statements “easier”
- lots of possible rules, different languages have different rules
- we define “strict” typing rules for HW-5 (more later)