Lecture 15:

- Semantic analysis (intro)

Announcements:

- HW-3 due Wed

Terms Relevant to Semantic Analysis

(1) **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)

(2) **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"

Terms Relevant to Semantic Analysis

(3) **Bindings**

- The association between names and objects ... name → object
- *Type bindings* connect names to their types
- *Location bindings* connect names to their locations in memory
- *Value bindings* connect names to their corresponding values

(4) **Environments** (aka **Contexts**)

- The current set of bindings of a program, statement, expression
- *Typing environments* give name → type bindings (currently "visible")
- Can also speak of environments at runtime (for locations and values)

Terms Relevant to Semantic Analysis

(5) **Scope Rules** (aka **Visibility Rules**)

- Specify what names 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 containing blocks

(6) **Static** and **Dynamic** ... review

- Static implies decisions made at *compile time* (before runtime)
- Dynamic implies decisions made at *runtime*

Terms Relevant to Semantic Analysis

(7) **Static Scope** (aka **Lexical Scope**)

- The visibility of names determined at compile time
- Based on the text of the source code
- What we normally think of as scope (visibility)

(8) **Dynamic Scope**

- The visibility of names determined at runtime
- Based on last association created for the name

(9) Most (modern) PLs primarily adopt static scoping rules

- some tricky cases though ...
- e.g., with nested functions, passing code blocks to functions, closures

Basic Semantic Analysis Examples

Programs can be syntactically correct but still have many errors

- **Goal**: Find and report errors statically (without running program code)

Detect type errors, e.g.:

```
x = 0 + "1"                    // int + string isn't allowed
if 42 <= true { ... }          // int <= bool isn't allowed
```

Detect "use before def" errors, e.g.:

```
var x: int = 42 + y            // y isn't defined
var y: int = x + f(x)          // f isn't defined
```

Detect function call errors, e.g.:

```
int add(x: int, y: int) {return x + y}
void main() {
  var r1: int = add(1, 2, 3);  // wrong number of args
  var r2: int = add(3.14, 1);  // wrong argument types
  var r3: bool = add(1, 2);    // wrong resulting type
}
```

and so on ...