

# Checkin 14

Does this program compile in a static scoping scheme? In a dynamic scoping scheme?  
What is the output for each scheme in which it compiles?

```
a : int;
v : () -> void {
    a = a + 1;
    consoleout a;
}
w : () -> void {
    a = 7;
    v();
    consoleout a;
}
main : () -> void {
    a = 0;
    a:int;
    a = 1;
    w();
    consoleoout a;
}
```

University of Kansas | Drew Davidson



*ECCS 665*  
**COMPILER**  
*CONSTRUCTION*

The title is presented in a highly stylized, futuristic font. The words "COMPILER" and "CONSTRUCTION" are in a bold, metallic, 3D style with a blue and white glow. "ECCS 665" is written in a pink, cursive script above "COMPILER". The background is a dark blue space with stars and a glowing purple grid floor.

Semantic Analysis

# Announcements

## Administrivia

# Announcements

## Administrivia

### P3 Questions

- The POSTDec operator
- The MAGIC operator
- Class-type variables and declarations

# Last Lecture

## Lecture Review – Scope

### Issues of Scope

- Scheme
- Shadowing
- Overloading

#### You Should Know

- Scope properties
- How scope affects semantics
- High-level scope rules for our language



# Lecture Outline

## Lecture Overview – Semantic Analysis

### **Name Analysis**

- Enforcing scope

### **Symbol Table**

- What it is
- What it does



**Semantics**

# Name Analysis

Semantic Analysis

## Idea:

- Associates IDs with their uses in the program
  - i.e. Emplace symbol table entries
- Implemented as an AST pass

## Purpose:

- Needed for code generation
- Catch some obvious errors
  - (undeclared IDs)



# Recognizing Identifier Context

Semantic Analysis

## A Context-Free Grammar is... context free

- A node needs information from outside of its subtree
- The definition of a identifiers needs to be connected to its occurrences
- We need a data structure to propagate such context



There's lots of ways to do this!  
We'll just cover 1 way

# The Compiler's Symbol Table

Semantic Analysis

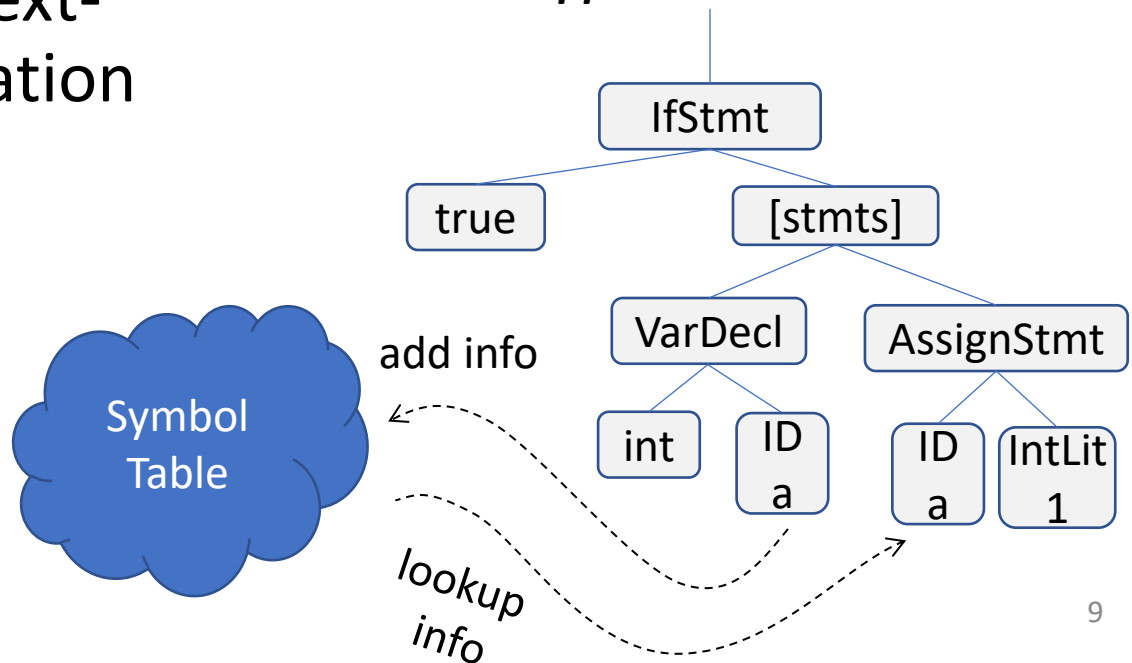
## Repository of semantic symbol information

- Populated during a walk of the AST
- Propagates context-sensitive information

### Program Snippet

```
if(true){  
    int a;  
    a = 1;  
}
```

### AST Snippet



# The Compiler's Symbol Table

## Semantic Analysis

### What's in the symbol table:

- Depends on the language

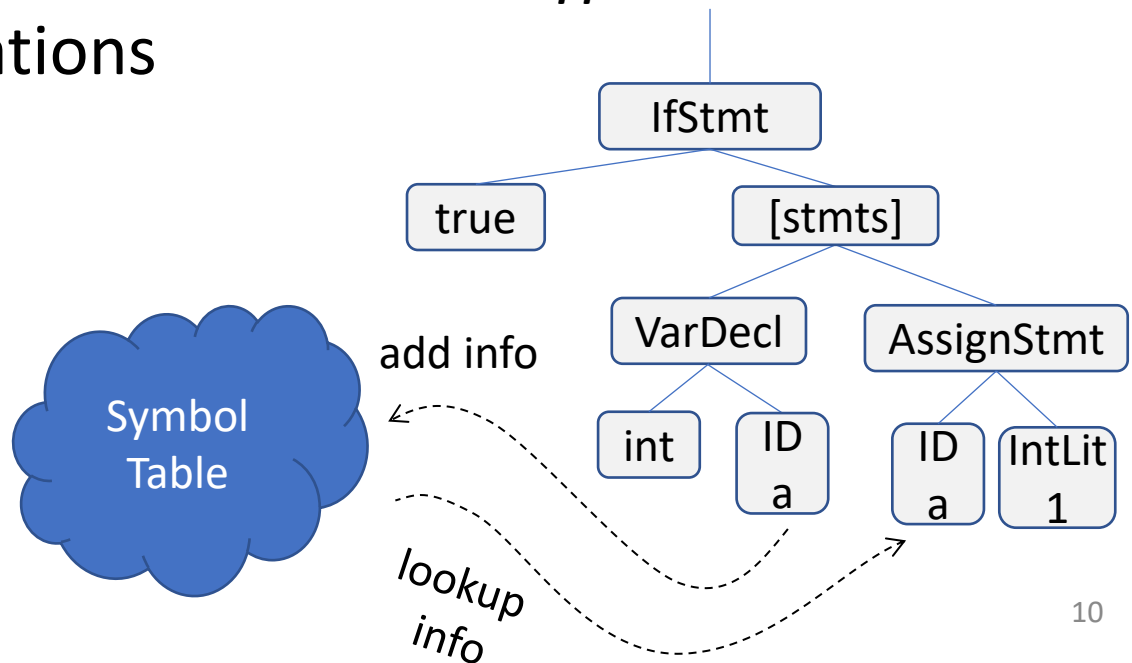
### Kinds of entries we need:

- Variable Declarations
- Function Declarations

### Program Snippet

```
if(true){  
    int a;  
    a = 1;  
}
```

### AST Snippet



# The Compiler's Symbol Table

Semantic Analysis

## What's in the symbol table:

- Depends on the language

## Kinds of entries we need:

- Variable Declarations
- Function Declarations
- Class Declarations

### Program Snippet

```
if(true){  
    int a;  
    a = 1;  
}
```

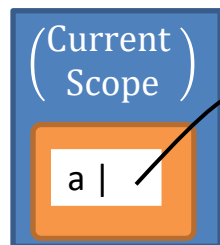
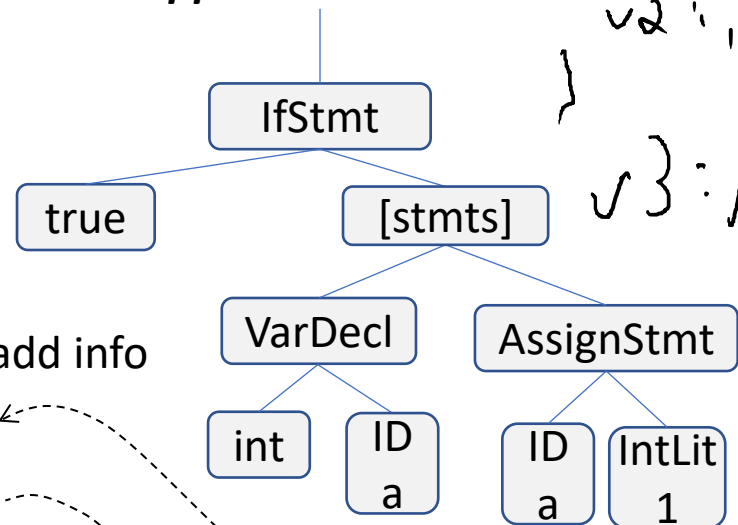
*var: int;*

*nm: () → void*

*custom M {  
 v1: int  
 v2: int  
}*

*v3: M;*

### AST Snippet



kind: var  
type: int

/add info

lookup  
info

# Symbol Table: A “Snapshot” of Scope

## Types – Name Analysis

### At any (static) program point

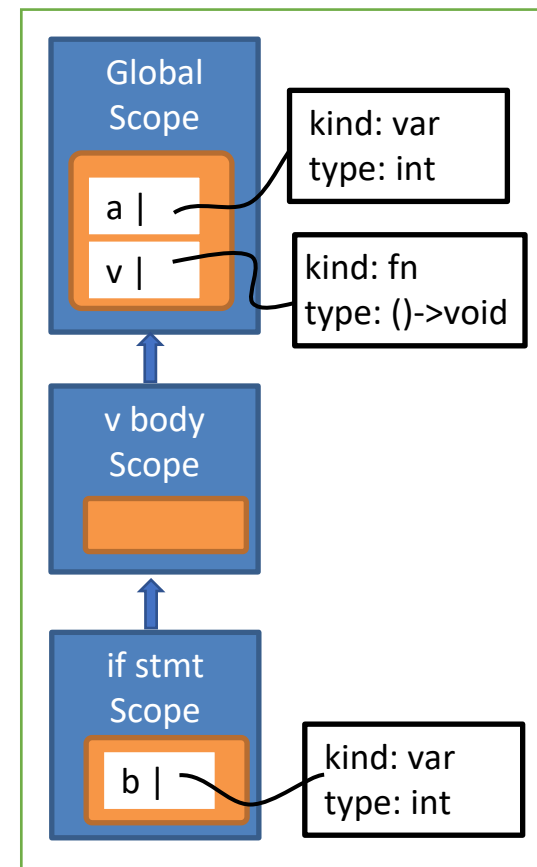
- Symbol table shows *what’s* in scope
- Symbol table shows which scope contains the entry

### Implementation:

- A list of hashmaps (1 map per scope)

```
1. int a;  
2. void v() {  
3.     if (a) {  
4.         int b;  
5.     }  
6. }
```

*Symbol table after line 4*

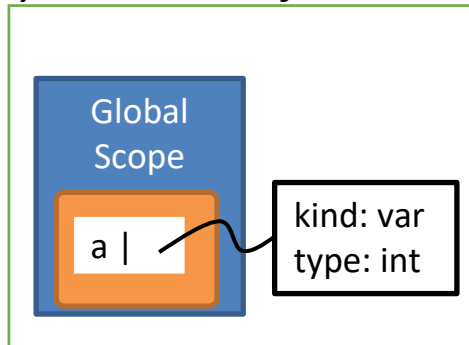


# Symbol Table: Scopes “Sub-tables”

Semantic Analysis - Name Analysis

## Create one hashmap per scope

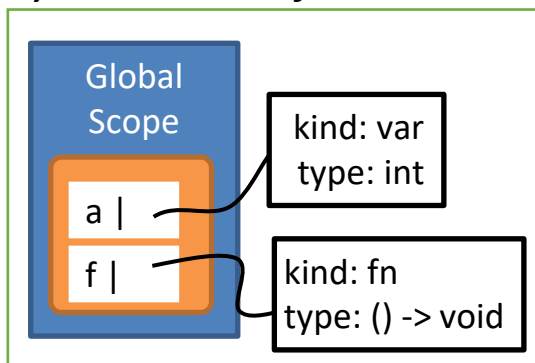
*Symbol table after line 1*



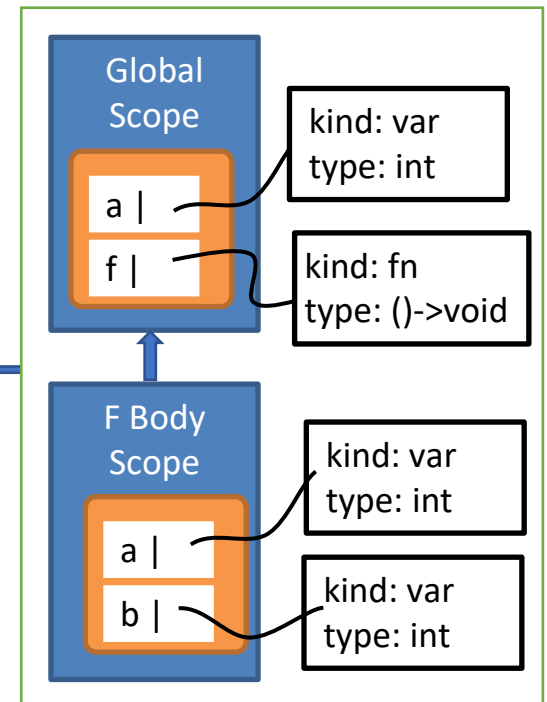
### Code

```
1. int a;  
2. void f() {  
3.     int a;  
4.     int b;  
5. }
```

*Symbol table after line 5*



*Symbol table after line 4*



# Implementation

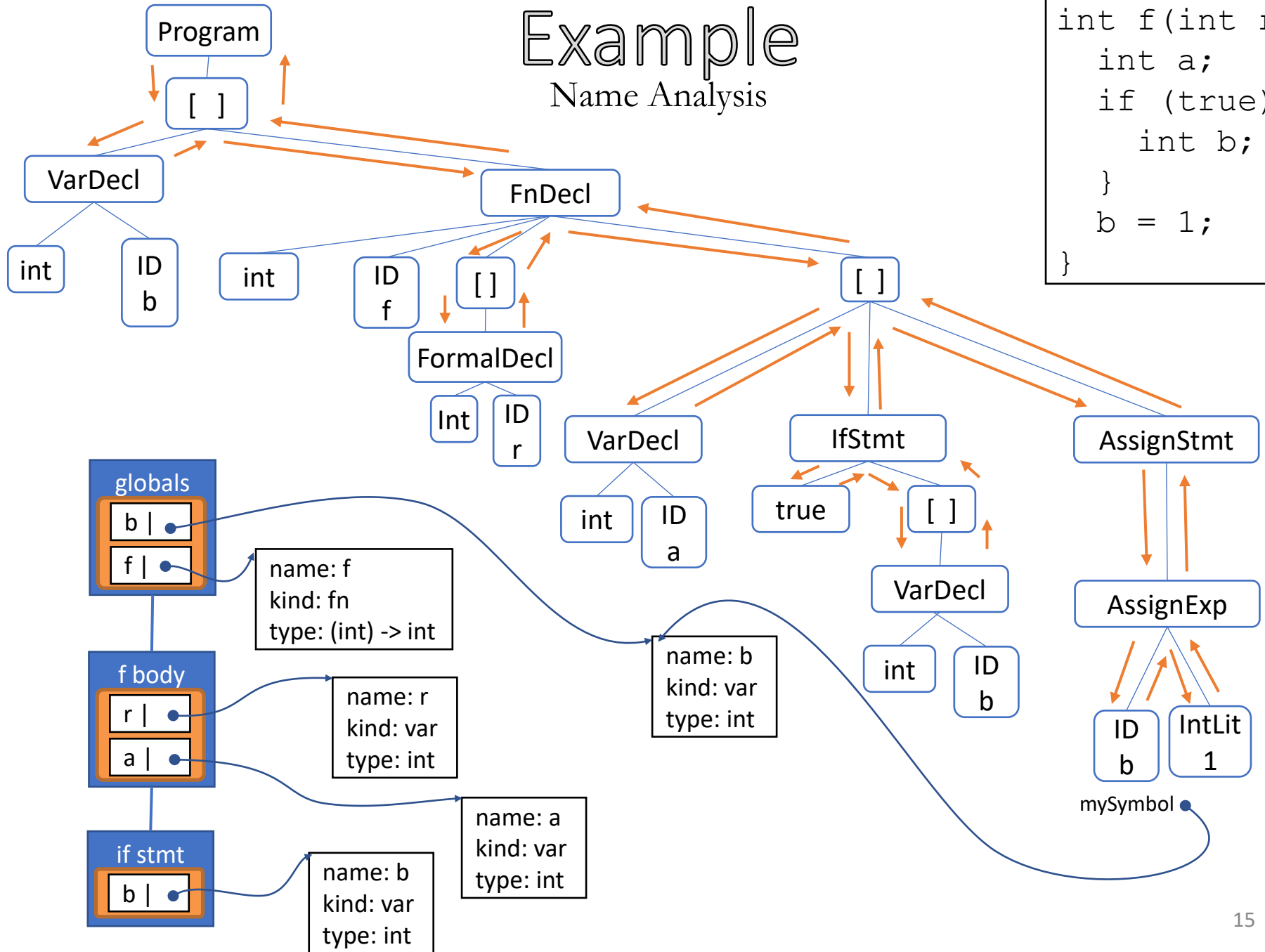
## Semantic Analysis - Name Analysis

- Walk the AST, much like the `unparse()` method
  - Augment AST nodes with a link to the relevant name in the symbol table
  - Build new entries into the symbol table when a declaration is encountered
  - Connect AST nodes to the entry they add or reference in the symbol table

# Example

## Name Analysis

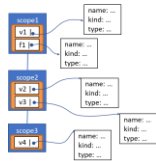
```
int b;
int f(int r){
    int a;
    if (true){
        int b;
    }
    b = 1;
}
```



# (My) Terminology

## Name Analysis - Implementation

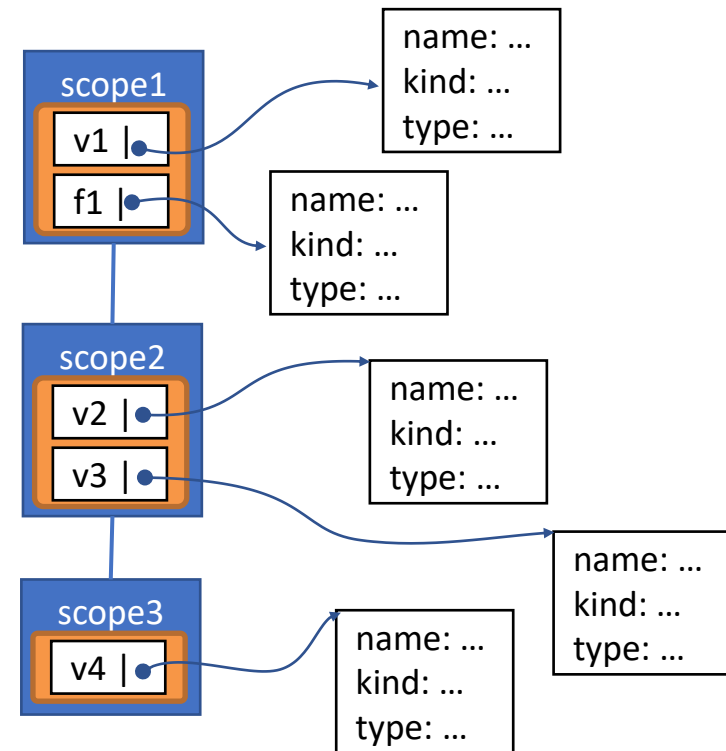
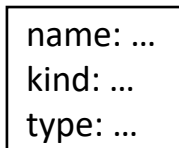
- Symbol Table – the whole structure



- Scope Table – A single map



- Symbol Table Entry (AKA “Semantic Symbol”)



# Implementation Suggestions

## Name Analysis - Implementation

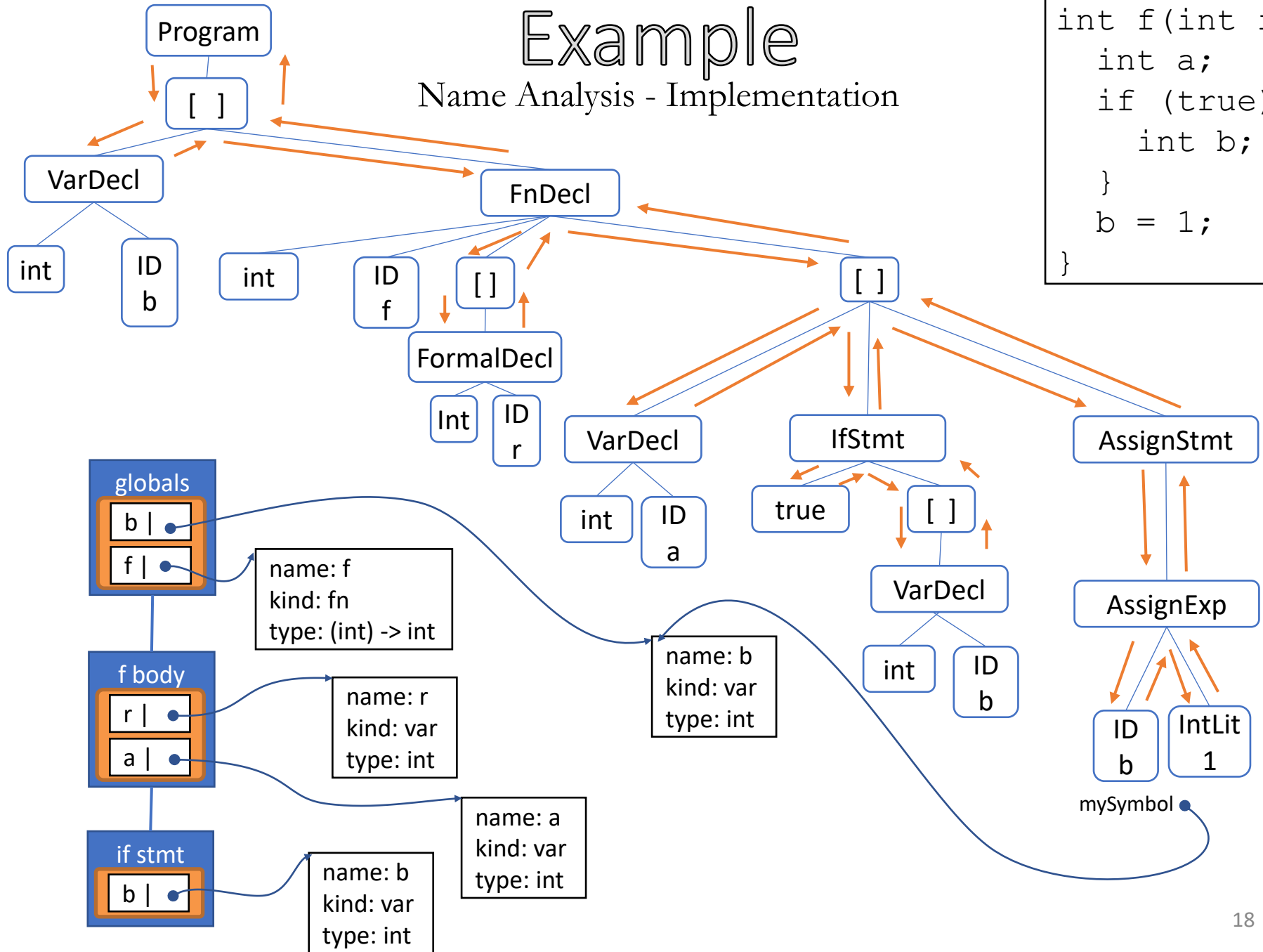
### Two approaches

- A `nameAnalysis` method for each `ASTNode` subclass
  - Override as appropriate
- The use of the visitor pattern

# Example

## Name Analysis - Implementation

```
int b;
int f(int r){
    int a;
    if (true){
        int b;
    }
    b = 1;
}
```



# Summary

## Name Analysis – Wrap-Up

- Described an analysis for enforcing static scoping
- Demonstrated a way to implement the analysis as a walk over the AST

# Next Time

## Name Analysis – Wrap-Up

### **Type Systems**

- What type systems are
- Why we use them
- The type system for our language

# Scratch Page

## Name Analysis – Scratch

# Scratch Page

## Name Analysis – Scratch

# Scratch Page

## Name Analysis – Scratch

# Scratch Page

## Name Analysis – Scratch

# Scratch Page

## Name Analysis – Scratch