CSE431 – Translation of Computer Languages

- Symbol Tables

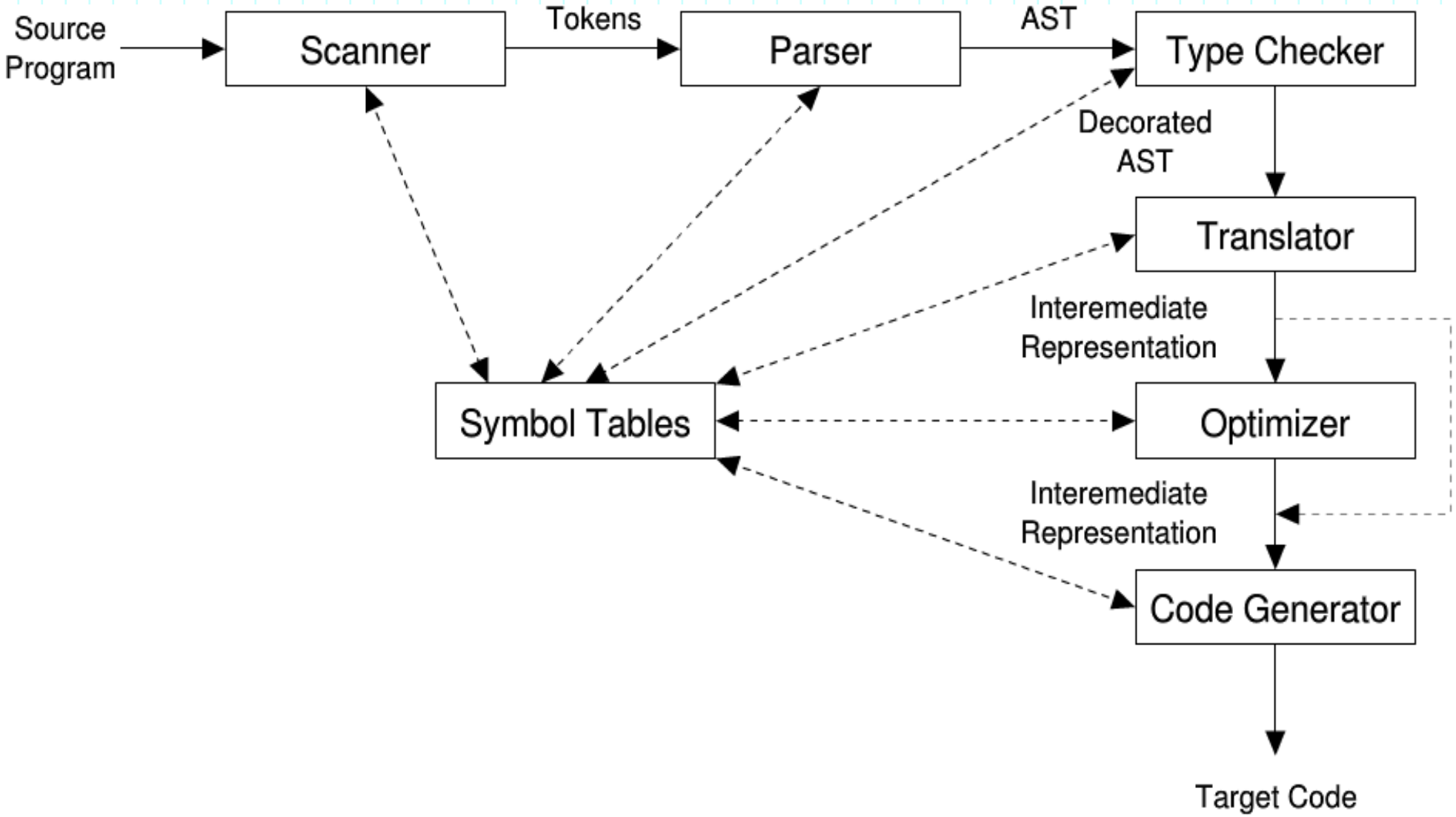
Doug Shook

Where are we?

- Scanning is complete
- Parsing is complete
  - Code is syntactically valid

- AST construction is complete
  - Forms the basis of semantic analysis
  - Remaining phases operate on this structure
    - Type checking is one such phase
      - Relies on a Symbol Table

Where are we?


Source Program
Scanner
Tokens
Parser
AST
Type Checker
Decorated AST
Translator
Interemediate Representation
Symbol Tables
Optimizer
Interemediate Representation
Code Generator
Target Code

Symbol Tables

- Used to store two primary pieces of information:
  - Symbols: how is this value being referred to in the code?
  - Type information
    - Scope
    - Initialization
    - Storage information
    - Type
- Two passes of the AST are required in this phase
  - Declarations
  - Usages
- Why not one pass?

Symbol Table Interface

- OpenScope()
  - New symbols added to this scope
- CloseScope()
  - Closes the current scope and moves to the outer scope
- EnterSymbol(*name*, type)
  - Enters *name* in the symbol table's current scope
- RetrieveSymbol(*name)*
  - Returns the currently valid declaration for *name*
  - Returns *null* if no declaration available

Scopes

- Scopes are nested
  - So how would you define...
    - Current scope?
    - Open scopes?
    - Closed scopes?

- Visibility rules for names:
  - Any name in the current scope or an open scope is accessible
  - If a name is declared in more than one open scope, resolve the name to the innermost declaration
  - New declarations can only be made in the current scope

Example

```
public class Car{
   int id;
   int color;
   int GetType() {
      String id;
   }
   public class Wheel {
      Object id;
      int GetType() {
         float id;
      }
   }
}
```

Implementation

- What structure seems appropriate for tracking scopes?

- Two possible approaches:
  - One symbol table for each scope
  - One single, global symbol table

Entering and Finding Names

- Which is more important: entry or retrieval?

- Possible implementations:
  - Unordered List
    - Store scope name (or depth) with the name
  - Ordered List
    - Ordered by what?
  - Binary Search Trees
    - Best case? Worst case?
  - Balanced Trees
    - Red-black, splay, etc. Run time?
  - Hash Tables
    - Most common (more later)

Entries

- An efficient symbol table will have the following entries:
  - Name: used to resolve hash table collisions
  - Type: any desired type information
  - Hash: pointers to items that hash to the same value (usually doubly linked)
  - Var: Reference to outer declaration with the same name (why do we need this?)
  - Level: List of symbols declared at same scope
  - Depth: nesting depth of symbol (scope level)

Implementation

- Two structures for the table:
  - Hash table
    - Name only
    - Used to find symbols with same hash value
  - Scope Display
    - Maintains a list of symbols active at each level
      - The *i*th entry references a list of symbols active at depth *i*

Efficient Insertion

- Lookup the name first
  - If symbol already exists and is defined at this scope, then what?
- Insert into the hash table
  - On collision, this new entry will become the head of the collision list
  - New entry's var points to old reference (different scope)
- Insert into Scope Display list
  - Prepend to the front of the Level list

Efficient Implementation

- Symbol Lookup
  - Simple hash table lookup
  - What if there's a collision?
- Entering a scope
  - Create a new entry in the Scope Display
  - What should this entry be initialized to?
- Exiting a scope
  - Remove each entry from the hash
  - If the entry has a var list, insert it into the hash
  - Do we need to do anything to Scope Display?

Exercises

- Sketch a symbol table for the following code:

```
public class A() {
    int x;
    void f(int m) {
      float x, y;
      ...
      { int i, j, x; }
    }
    int g(int n) {
      bool t;
      ...
    }
}
```

For next time....

- Read section 8.3!
  - Pseudocode on page 292
    - Might ask you to implement this in studio....
    - Might be a contest involved...