EECS483 D6: Project 3 Overview

Chun-Hung Hsiao

Feb 15, 2013

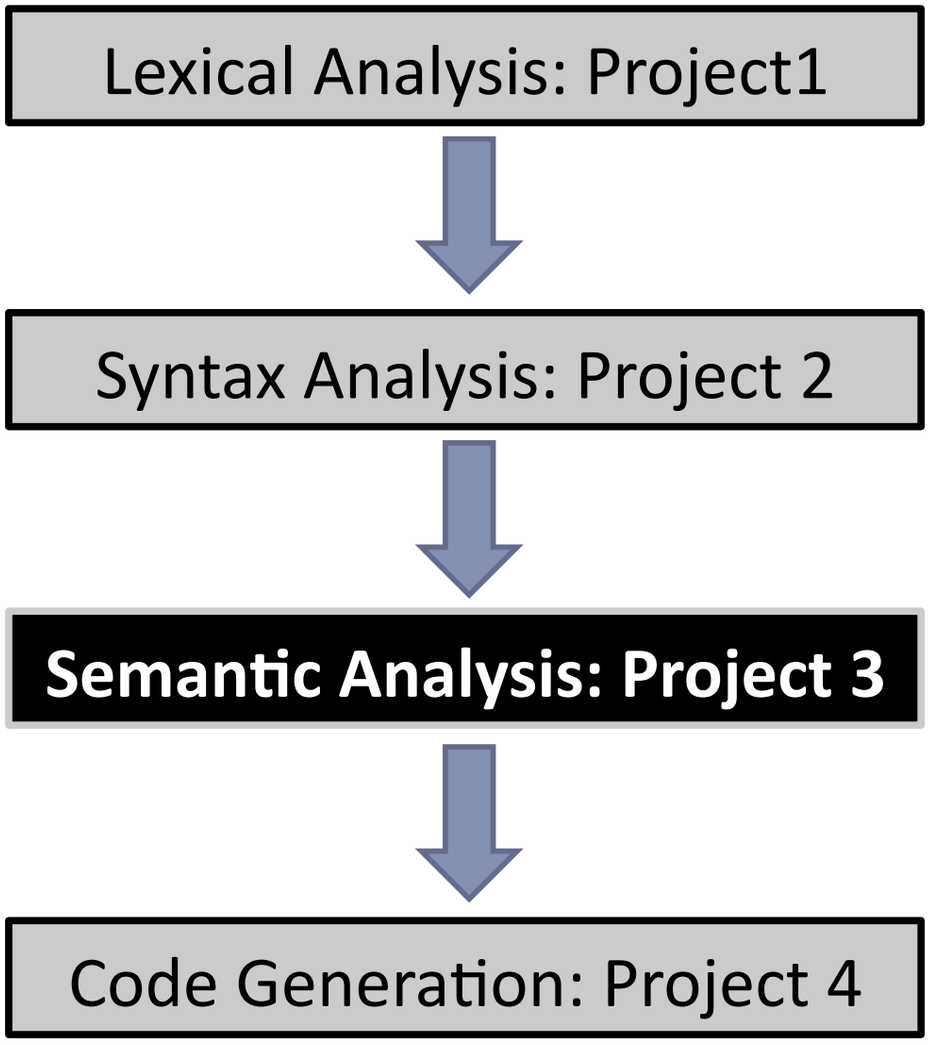
Special thanks to Ashutosh

Announcement

- We do have a discussion session on 2/22 (schedule updated)
- Homework 3 due on the next discussion session
- Project 3 announced and due on 3/18
  - Checkpoint on 3/11
- Project 3 checkpoint policy updated

Project 3 Overview

- Project divided into two stages
  - Checkpoint on 3/11
  - Final submission on 3/18
- Objective: Semantic analysis
  - Locate semantic errors in Decaf
  - Just one step away from code generation!
- Not a one-night shot!



Checkpoint Policy

- To make sure that you start the project early
- The checkpoint worth 10 extra bonus points
- Will test your submissions on a smaller test set
  - 10 points for passing 20 tests or more
  - 5 points for passing 10 tests or more
  - 0 point otherwise
- No late day for checkpoint
- Use project number “3c” before checkpoint; use “3” after that
  - You need to submit your code after checkpoint to get the full credits

Infrastructure

- Classes for AST nodes are the same as in PP2
  - The printing functionality is removed
- Replace parser.y with your own one in PP2
  - Call program->Check() instead of program->Print()
  - Make sure that you use the correct location information when allocating a new AST node
  - You can use the sample solution after we release it
- A sample dcc provided in the solution/ folder
  - Construct your own test case and use it to generate a reference output

What to Do for Checkpoint

- Read the semantic rules of Decaf carefully
- Scope system: design strategy for detecting scopes
  - What info needs to be recorded with each scope?
  - How will you store the scope info?
  - What is the rules of scope visibility?
- Type system: type inference and type checking
  - What is the type of the result of an expression?
  - What types are allowed in the context?
- Report errors when the semantic rules are violated

Error Reporting

- No output if there is no semantic error
- In case of semantic error
  - Report the line number of the error
  - Print a string describing the error
- You don't need to prepare the output all on your own
  - Line numbers are tracked when constructing the AST
  - Error strings are defined in errors.h/.cc
  - Use the provided ReportError library to print the error
- Your job is to call the correct functions corresponding to the errors discovered

Errors to be Reported at Checkpoint

- Conflicting declarations
- Undeclared identifiers
- Incomplete implementations
- Invalid self-references
- Invalid use of arrays

Conflicting Declarations

- ReportError::DeclConflict()

```
*** Declaration of 'a' here conflicts with declaration on line 5
```

  - Redeclaring a variable/function/class/interface
  - Formal parameters of the same name

- ReportError::OverrideMismatch()

```
*** Method 'b' must match inherited type signature
```

  - Overriding a method with a different type signature

Undeclared Identifiers

- ReportError::IdentifierNotDeclared()

```
*** No declaration for class 'Cow' found

*** No declaration for function 'Binky' found
```

  - Using a variable/function/class/interface without declaration

Incomplete Implementations

- ReportError::InterfaceNotImplemented()

```
*** Class 'Cow' does not implement entire interface 'Printable'
```

  - Missing a method listed in the interface that a class is implementing

Invalid Self-references

- ReportError::ThisOutsideClassScope()

```
*** 'this' is only valid within class scope
```

  - Using the “this” keyword outside a class method

Invalid Use of Arrays

- ReportError::BracketsOnNonArray()

```
*** [] can only be applied to arrays
```

  - Using [] operator on a non-array variable/expression

- ReportError::SubscriptNotInteger()

```
*** Array subscript must be an integer
```

  - Accessing an array element with a non-integer index

- ReportError::NewArraySizeNotInteger()

```
*** Size for NewArray must be an integer
```

  - Allocating an array with a non-integer size

Scope System

- What are different kinds of scopes?
- What to record within each scope?
- How to record a scope?
  - You can use the provided Hashtable library to map identifiers to their declarations
- How to lookup an identifier in the scope system?
- Do different kinds of scopes need special handling?

Type System

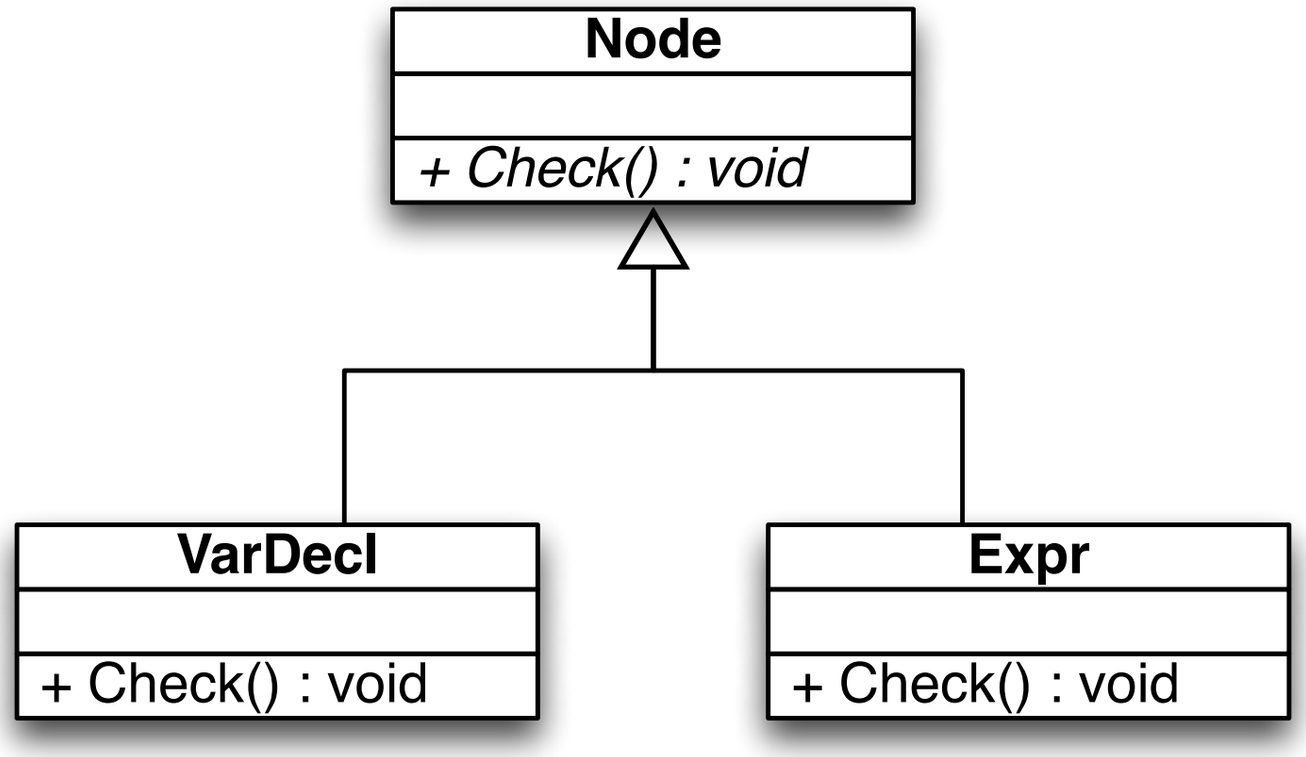
- How to get the type of an identifier?
- How to know the type of the result of an expression
- How to check if the type of a declaration or an expression is allowed in the context?
  - Type equivalence
  - Type compatibility (not needed at checkpoint)

Implementing Semantic Analysis

- Two approaches: 1-pass or 2-pass
- 1-pass approach
  - Reporting errors when parsing the input program
  - Implemented in the actions in parser.y, thus highly coupled with the parser code
  - Fast and memory-efficient compilation, but hard to implement certain features of Decaf
- 2-pass approach
  - Reporting errors by examining the AST after the input program is completely parsed
  - Implemented in the AST nodes
  - Suggested approach for PP3

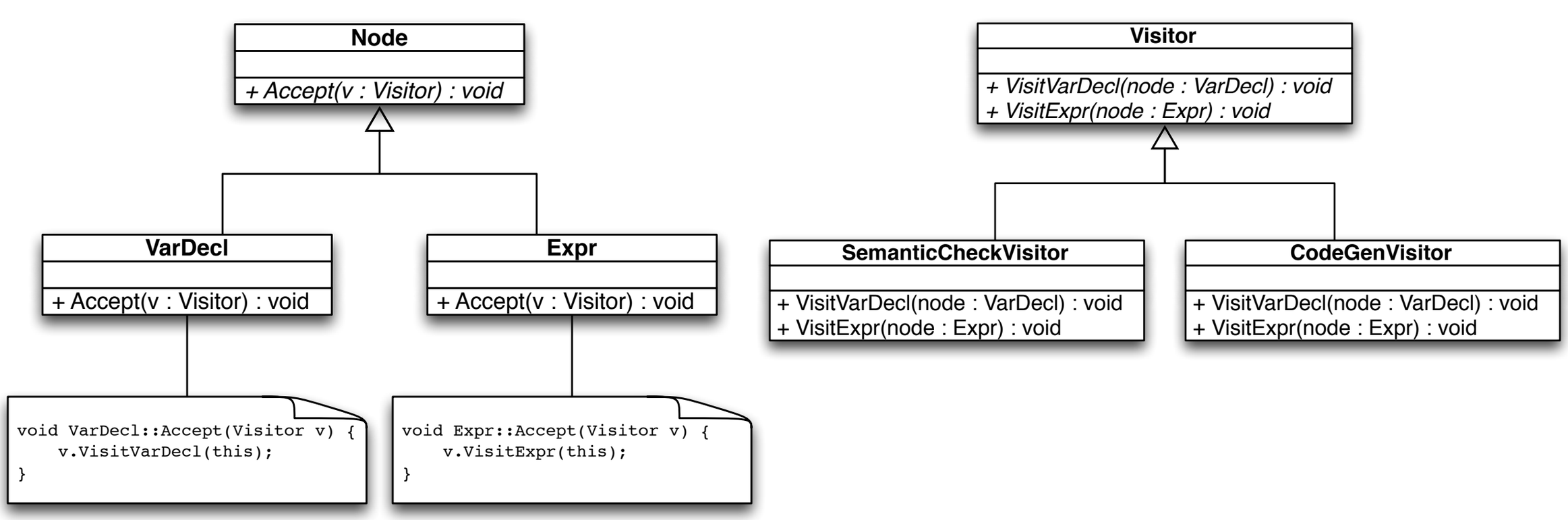
Method 1: Polymorphic Node::Check()

- Implement a polymorphic Check() function for each AST node
  - Maintaining scope and type information
  - Recursively check every child node
  - Report errors if the check failed



Method 2: Visitor Design Pattern

- The Visitor pattern is a perfect fit for developing a compiler
  - Decoupling the checking code from the AST nodes
  - More extensible in software engineering



Beyond the Checkpoint

- Implementing type compatibility checking
- Scoping rule for the “.” operator
- Handling cascading errors

Thanks & all the best!