EECS483 D8: Project 3 Details

Mar 1, 2013

Chun-Hung Hsiao

Reminder

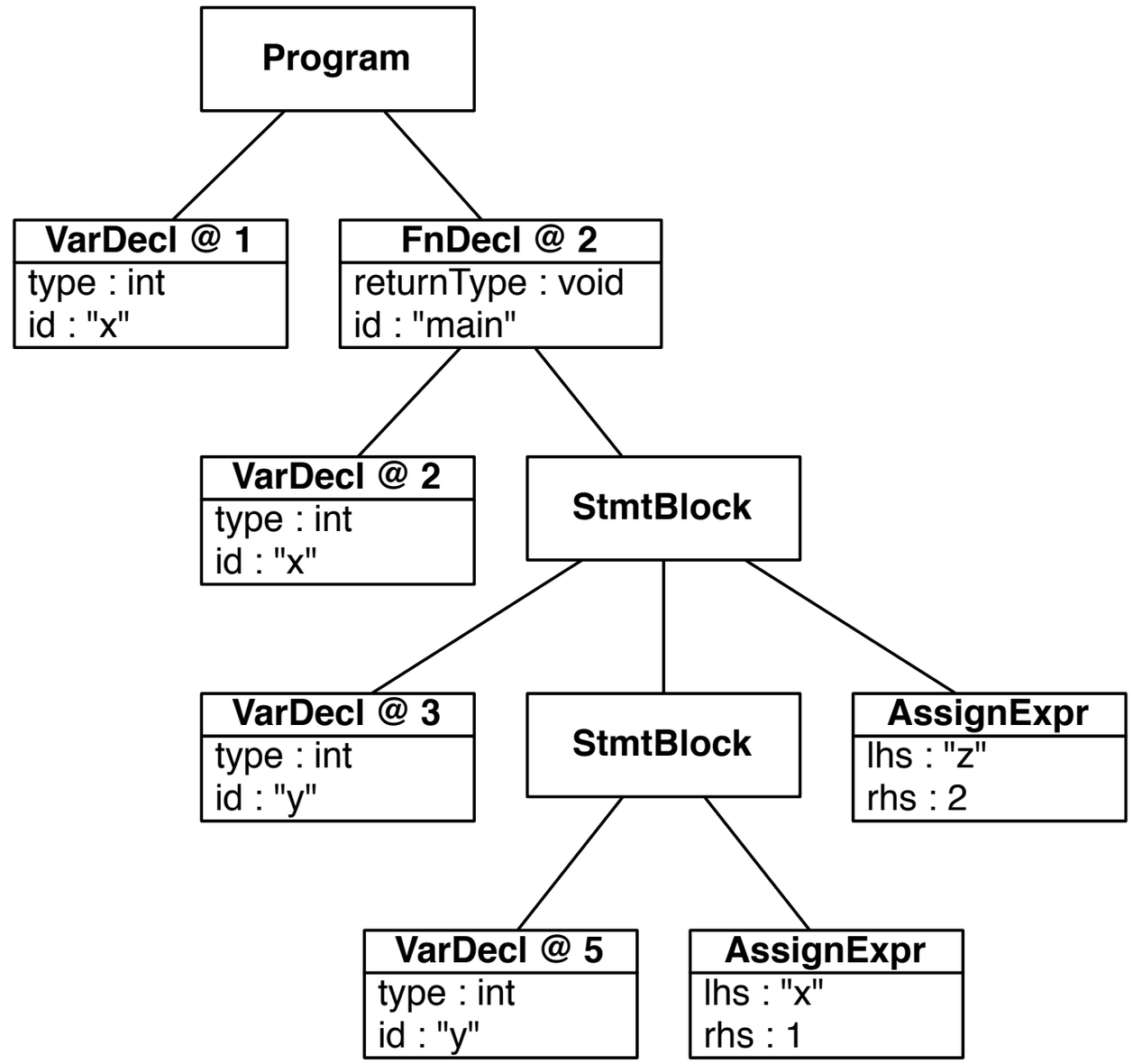
- Project 3 Checkpoint Due on 3/11
  - Get 5/10 bonus points if passed >= 10/20 tests
  - No late submission
  - Use “3c” as project number to submit
- Project 3 Due on 3/18
  - Use “3” as project number to submit
  - Open on 3/12
  - Will use a new test set
  - Remember to submit it again even if you have submitted the checkpoint

Implementing Scope System

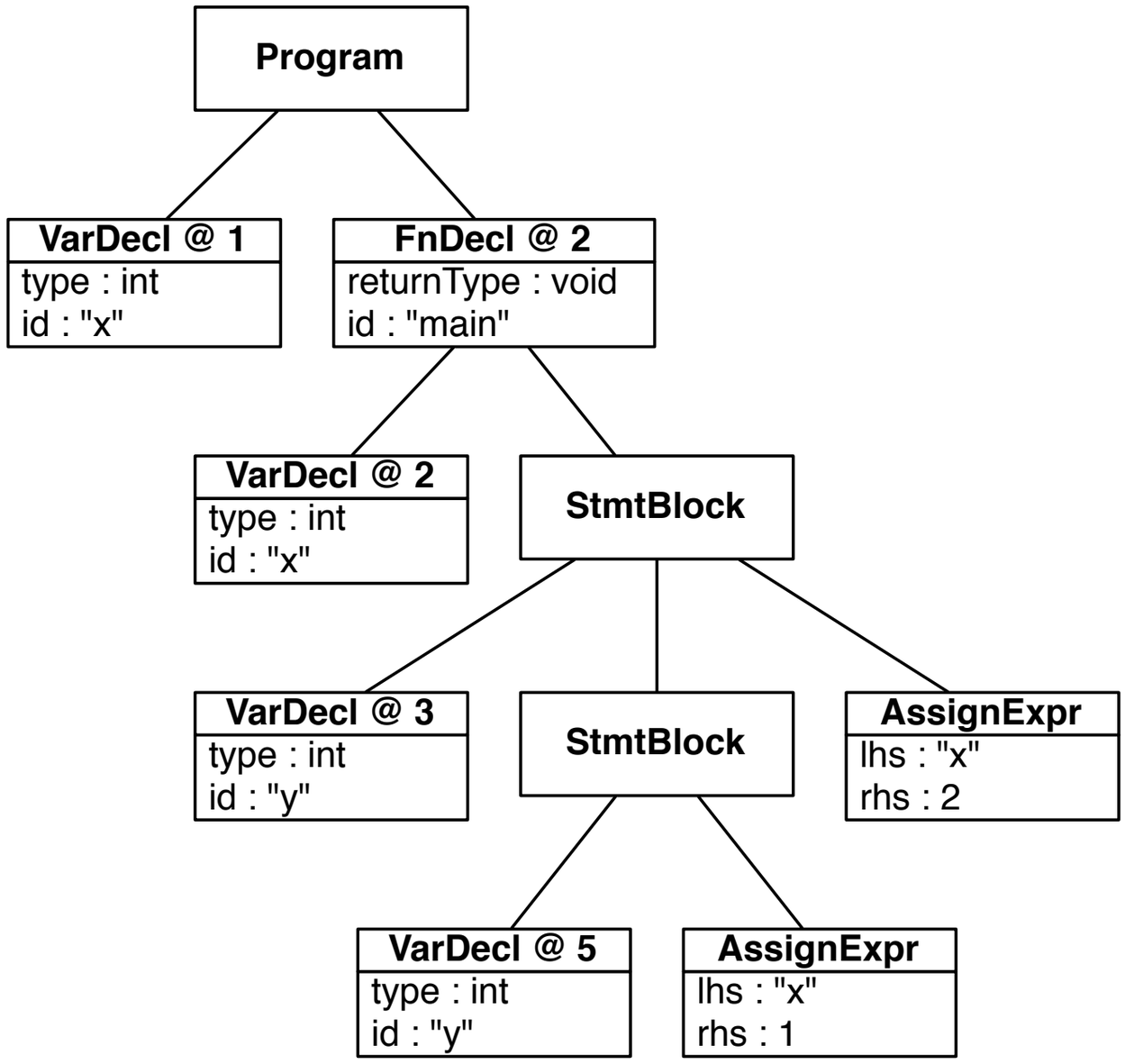
- Strategy: polymorphic Node::Check()
- Functionalities of Node::Check()
  - Check semantic errors in this node
  - Iteratively call Check() to traverse all children nodes to check their errors
  - Post-traversal checking
- For nodes that open new scopes
  - Maintain the stack of the symbol tables
- For declarations:
  - Put the symbol into the current symbol table

Recap: Abstract Syntax Tree

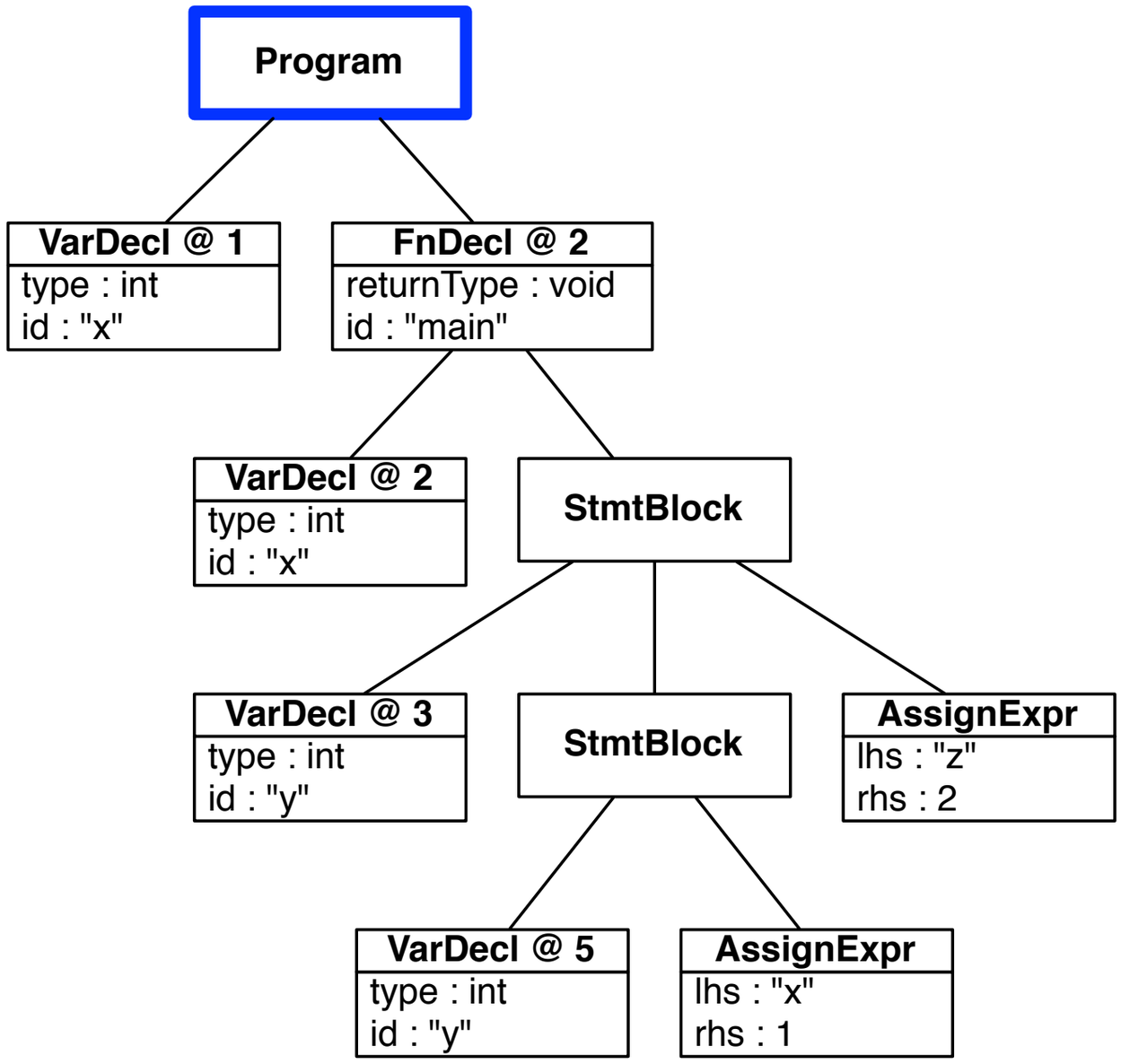
```
1 int x;
2 void f(int x) {
3       int y;
4       {
5           int y;
6           x = 1;
7       }
8       z = 2;
9 }
```



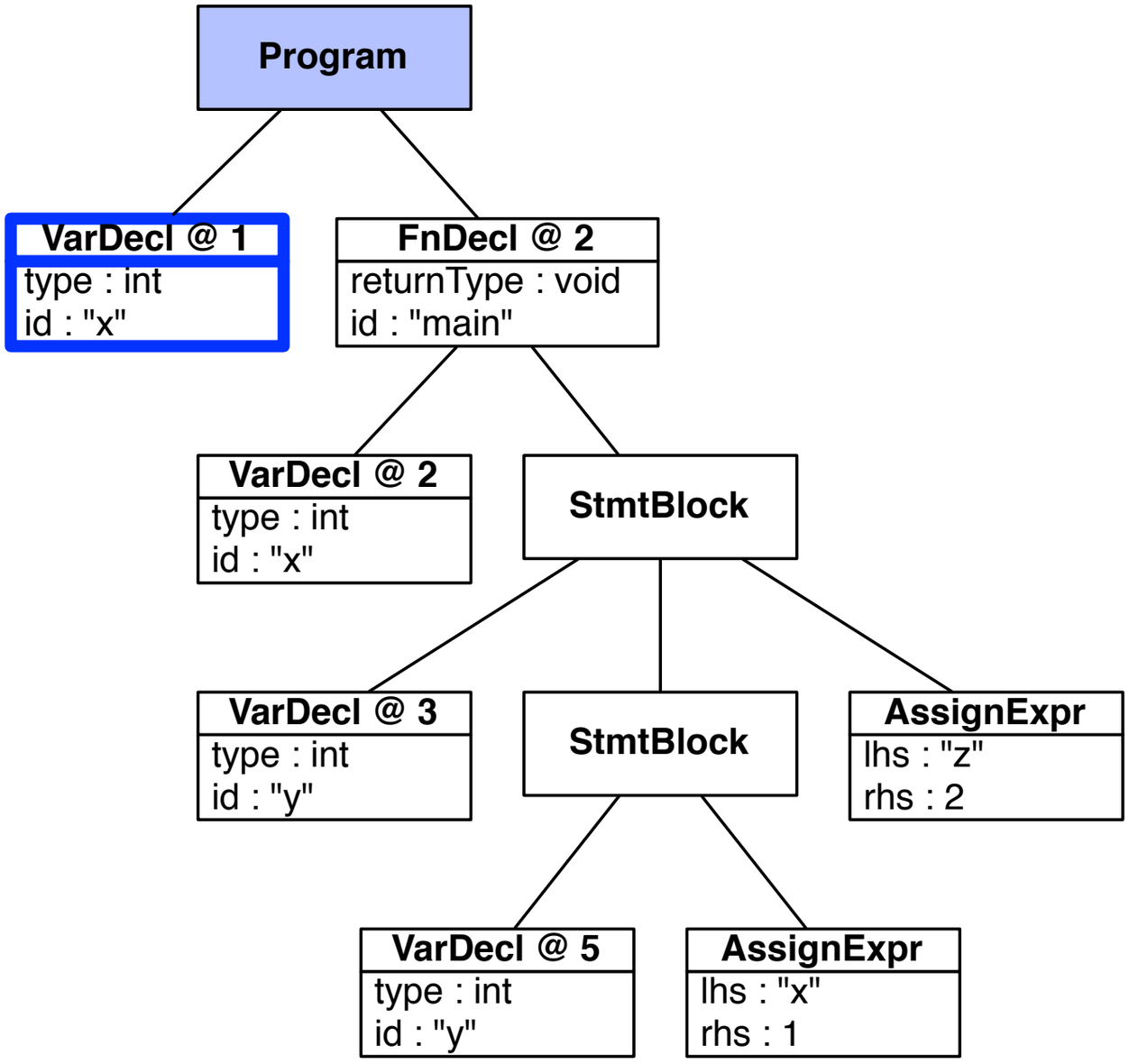
Scope System: Using Symbol Table



Scope System: Using Symbol Table

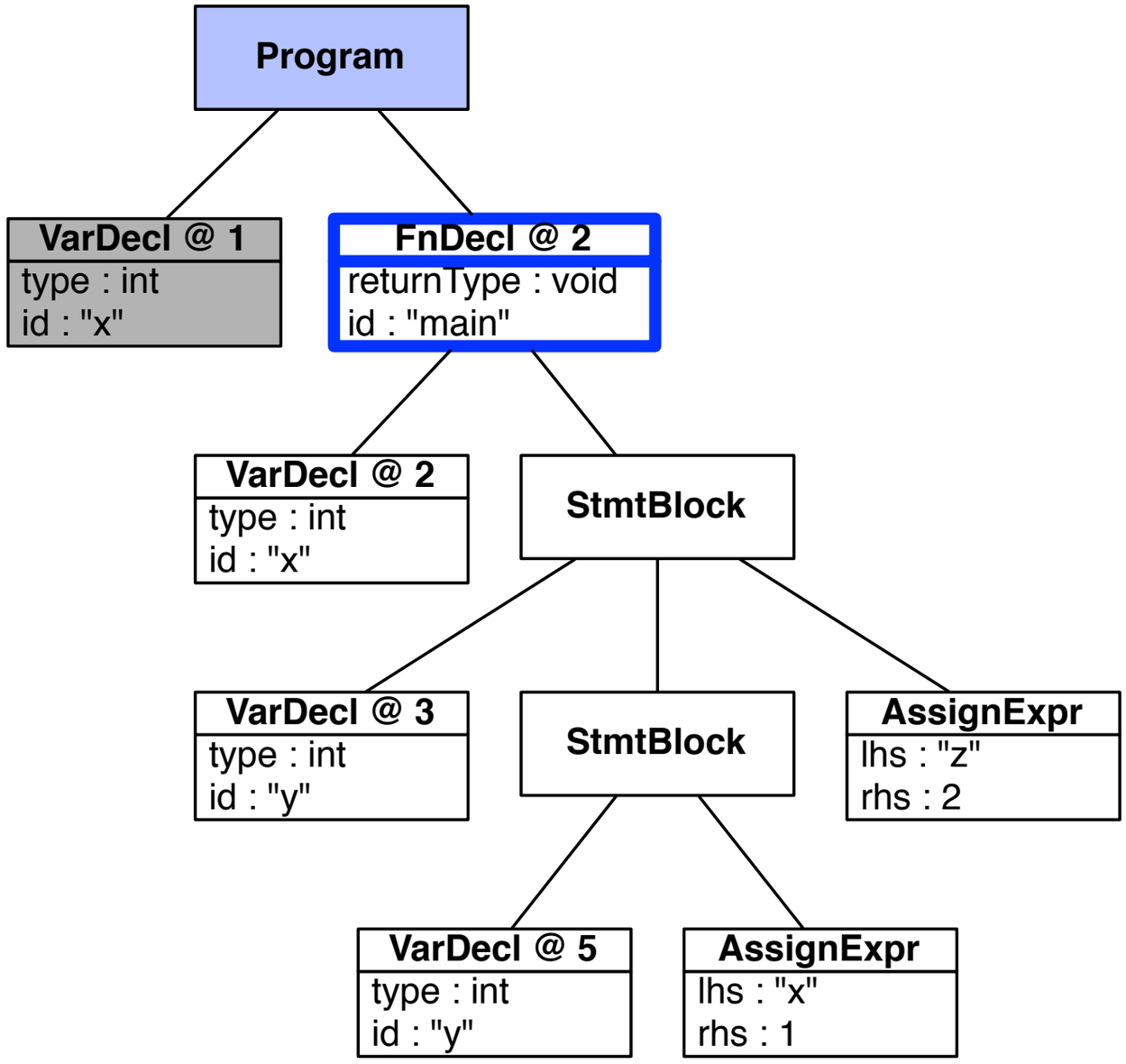


Scope System: Using Symbol Table



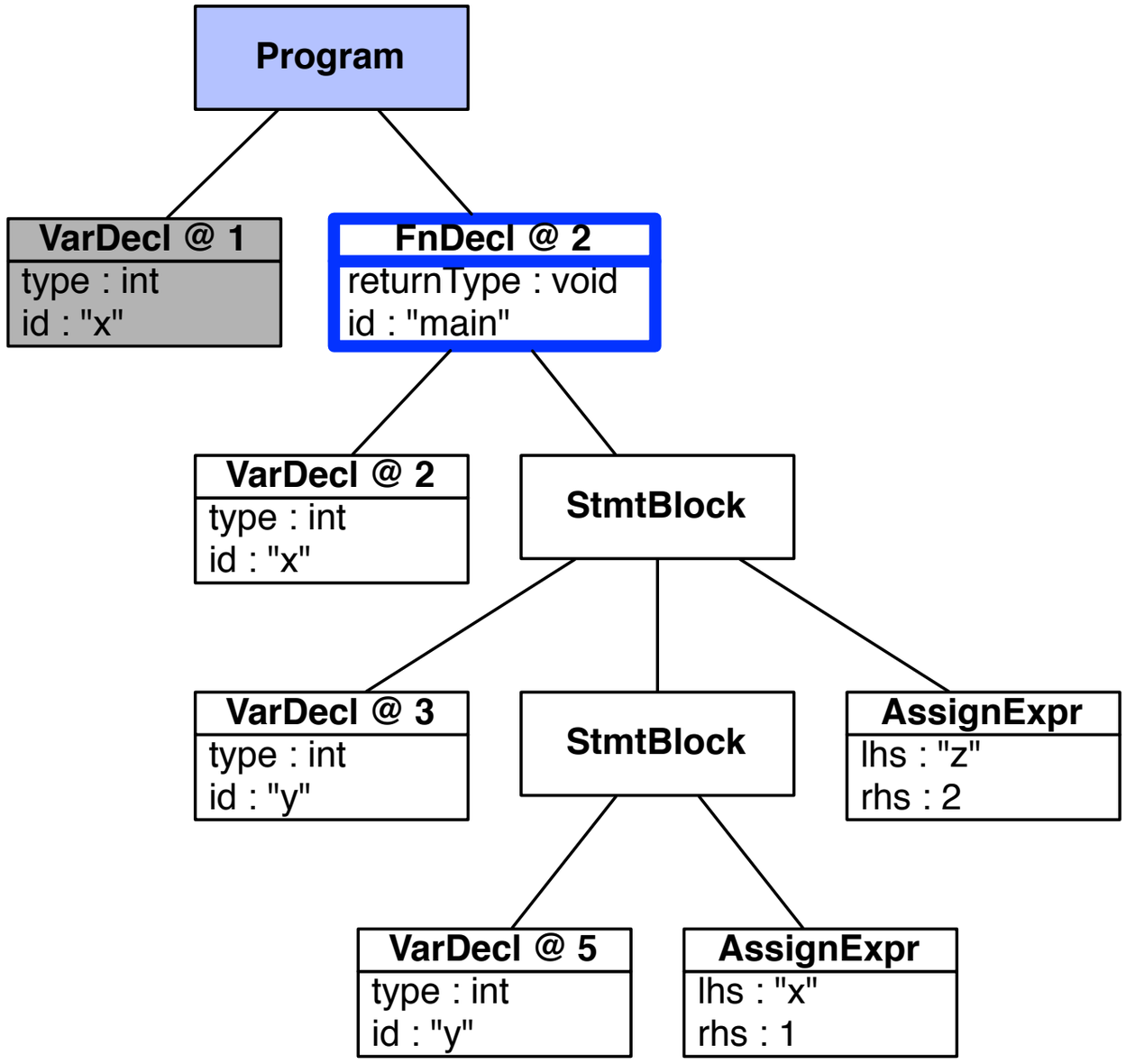
Symbol Table	
x	: VarDecl @ 1

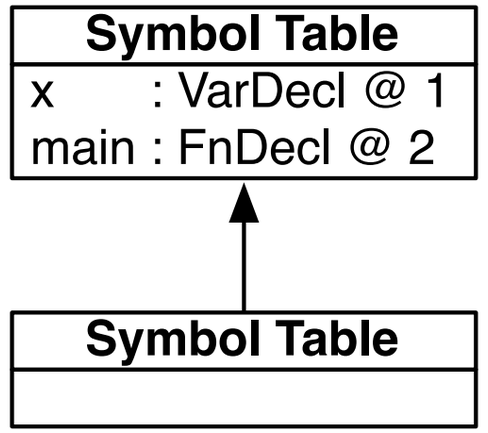
Scope System: Using Symbol Table



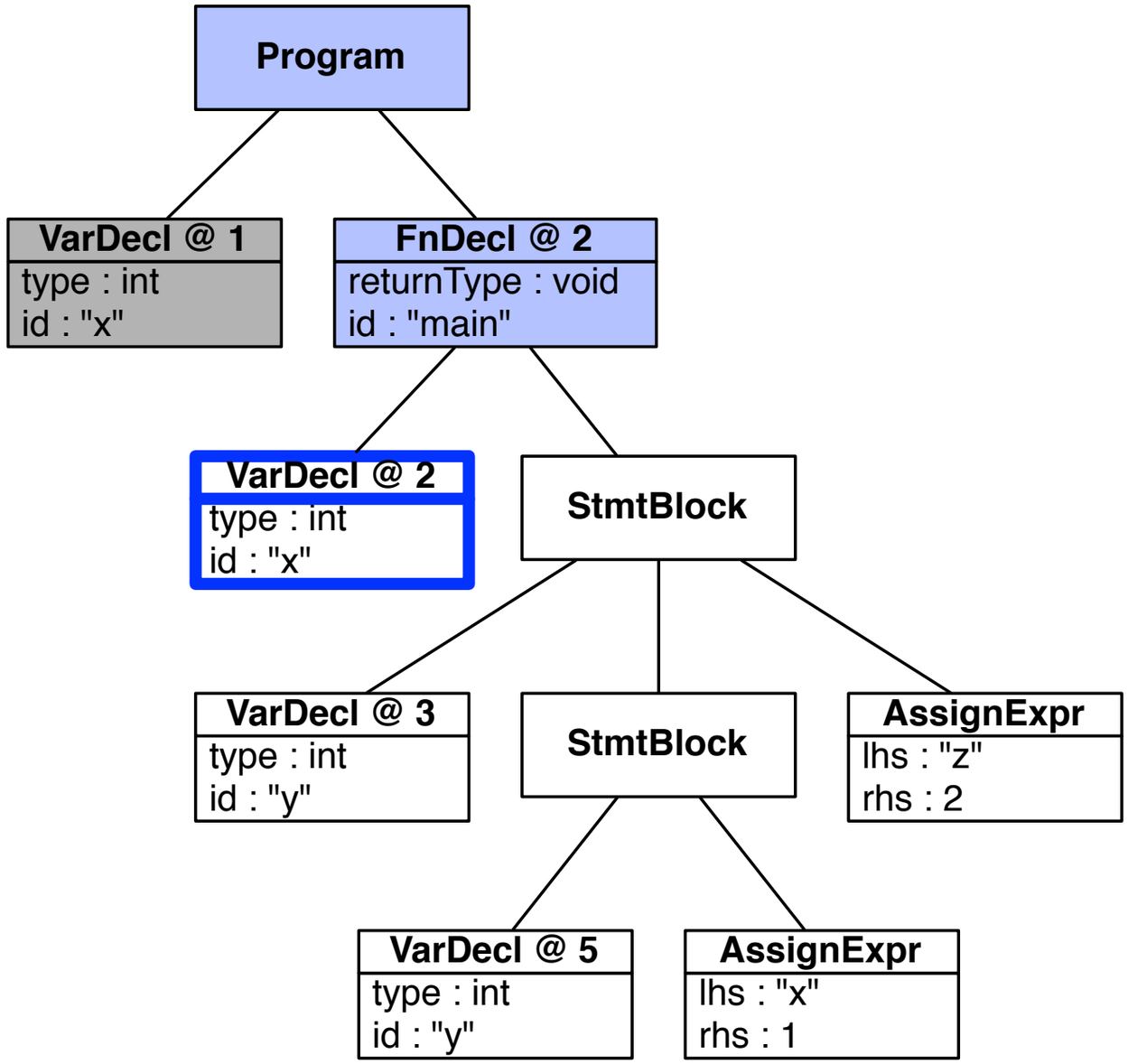
Symbol Table
x : VarDecl @ 1
main : FnDecl @ 2

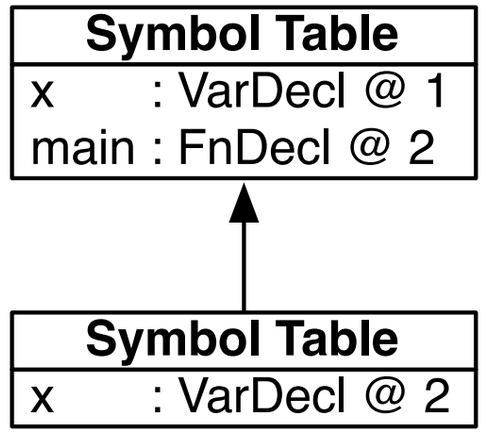
Scope System: Using Symbol Table



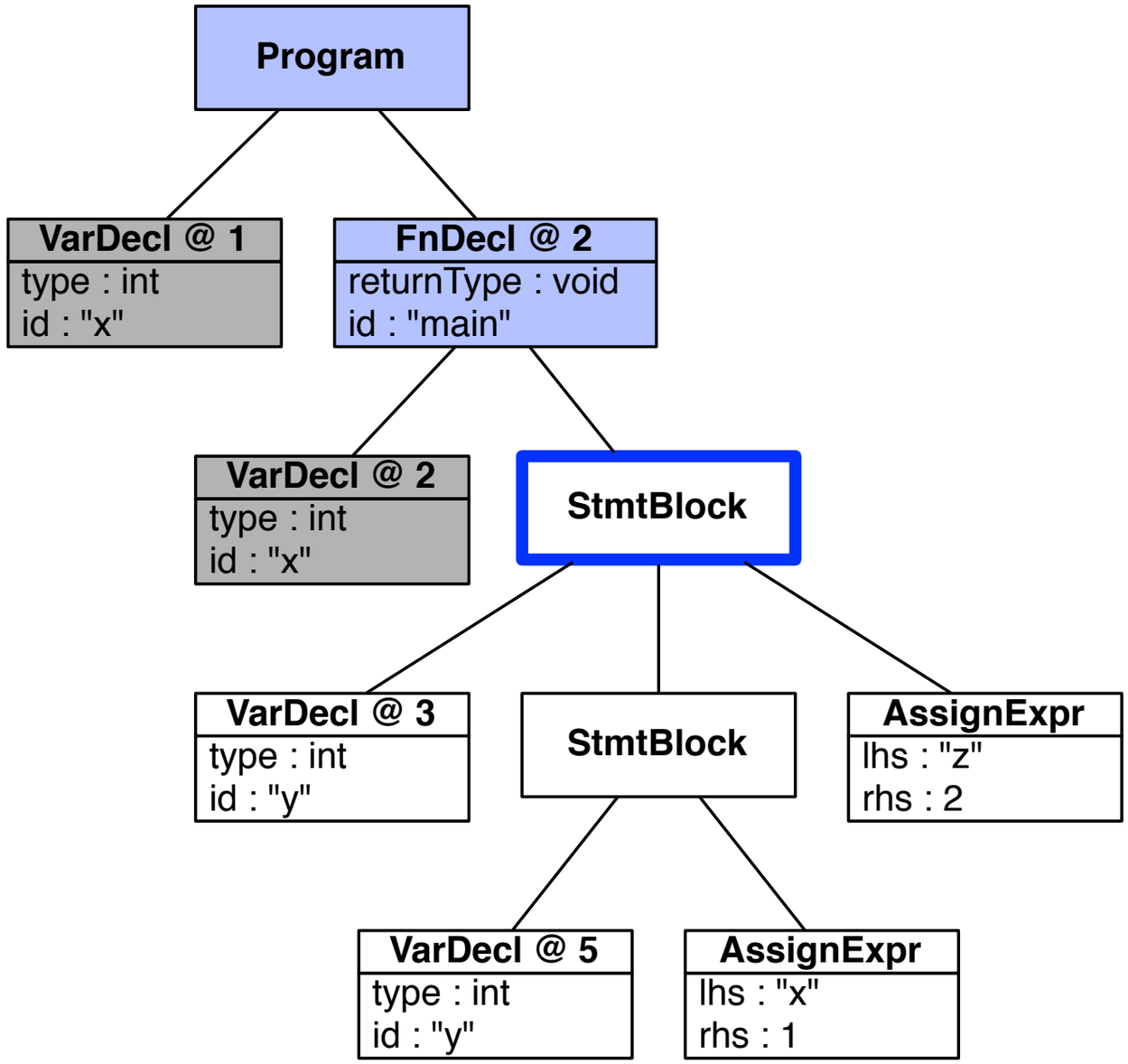


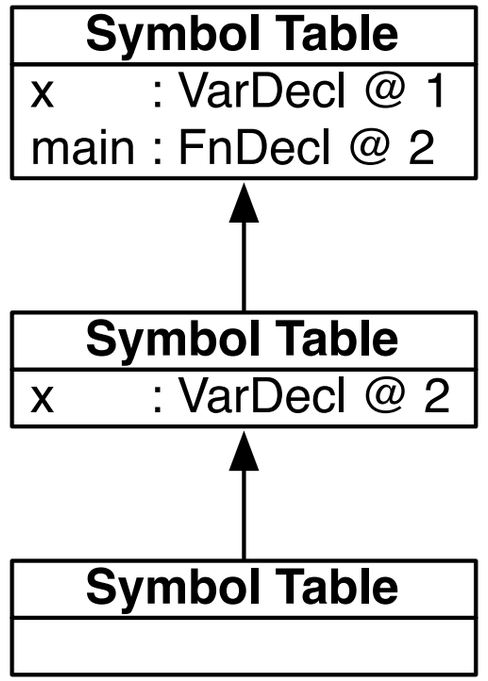
Scope System: Using Symbol Table



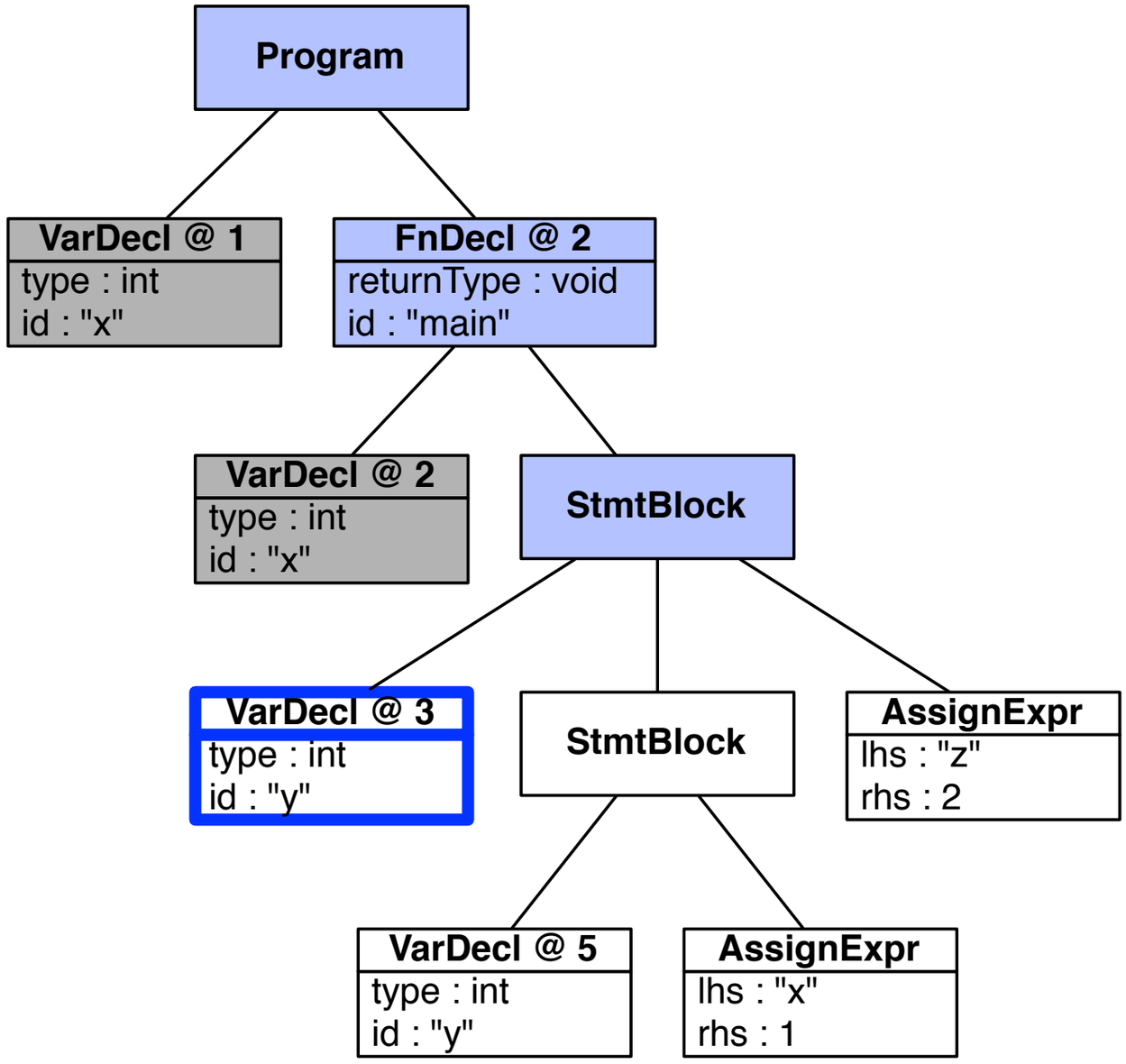


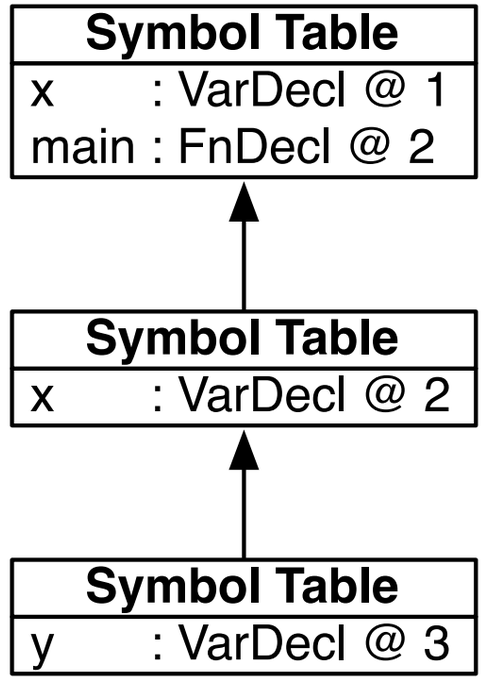
Scope System: Using Symbol Table



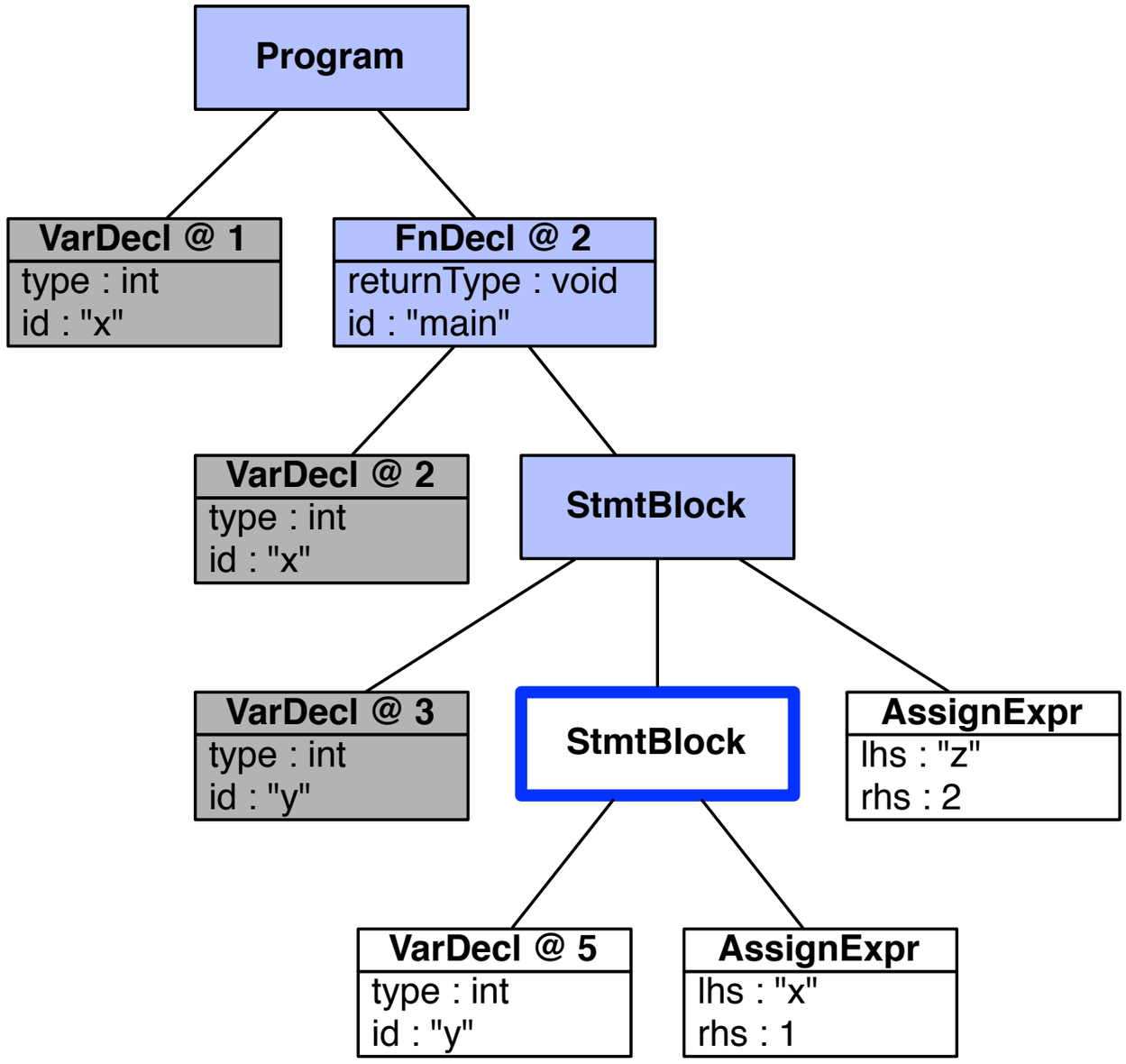


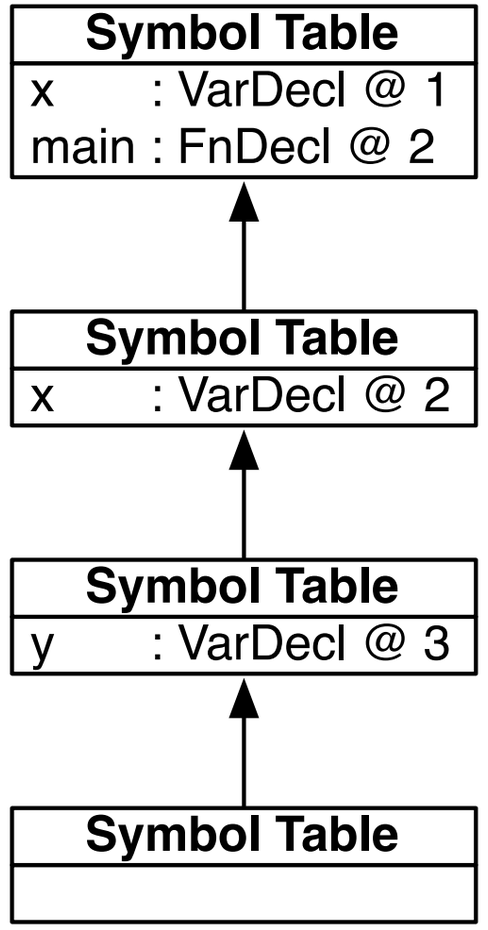
Scope System: Using Symbol Table



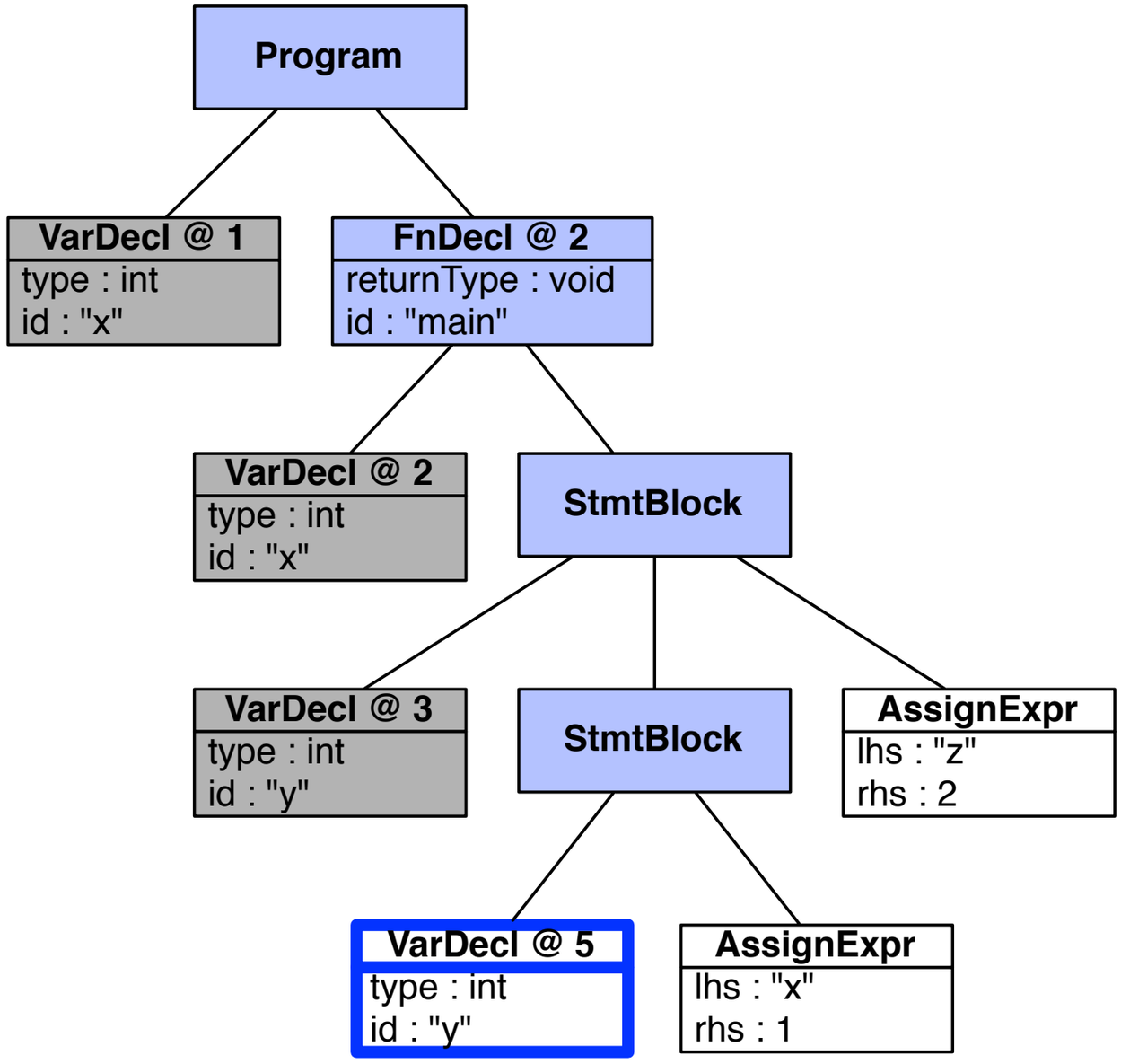


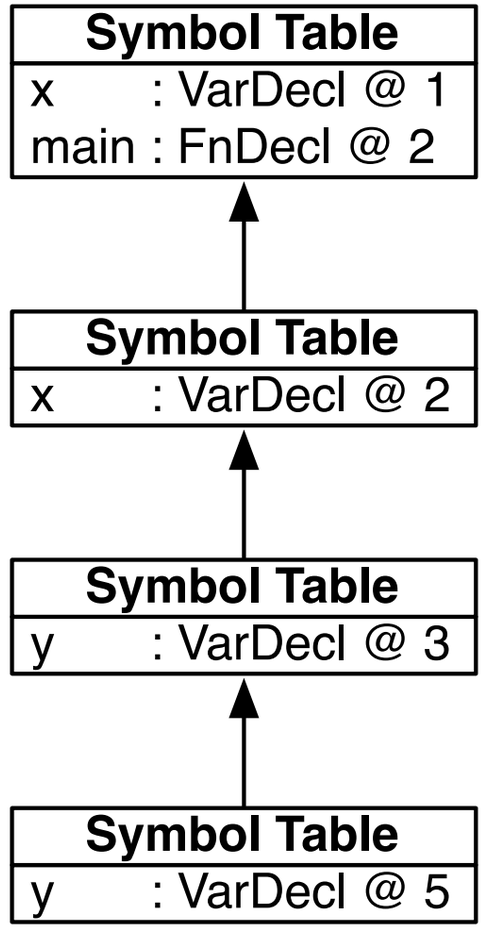
Scope System: Using Symbol Table



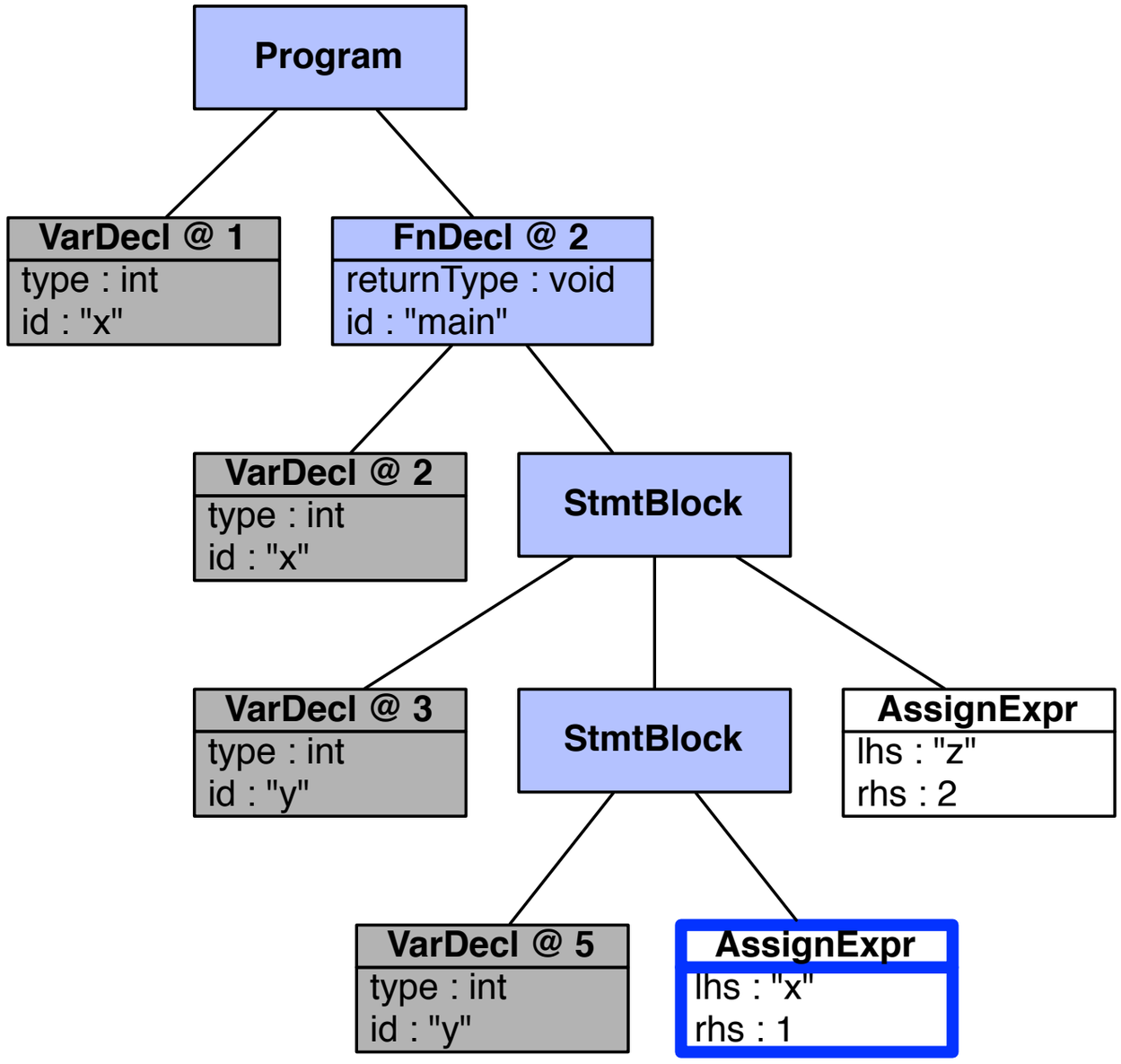


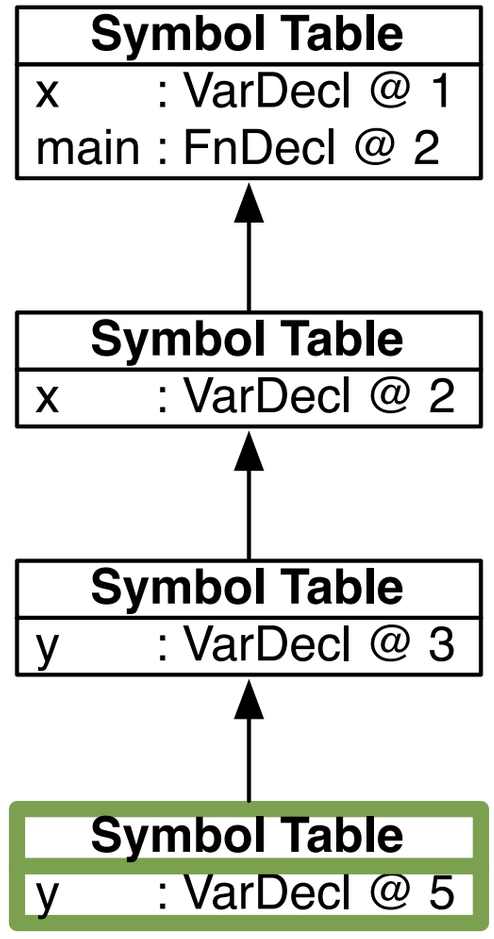
Scope System: Using Symbol Table



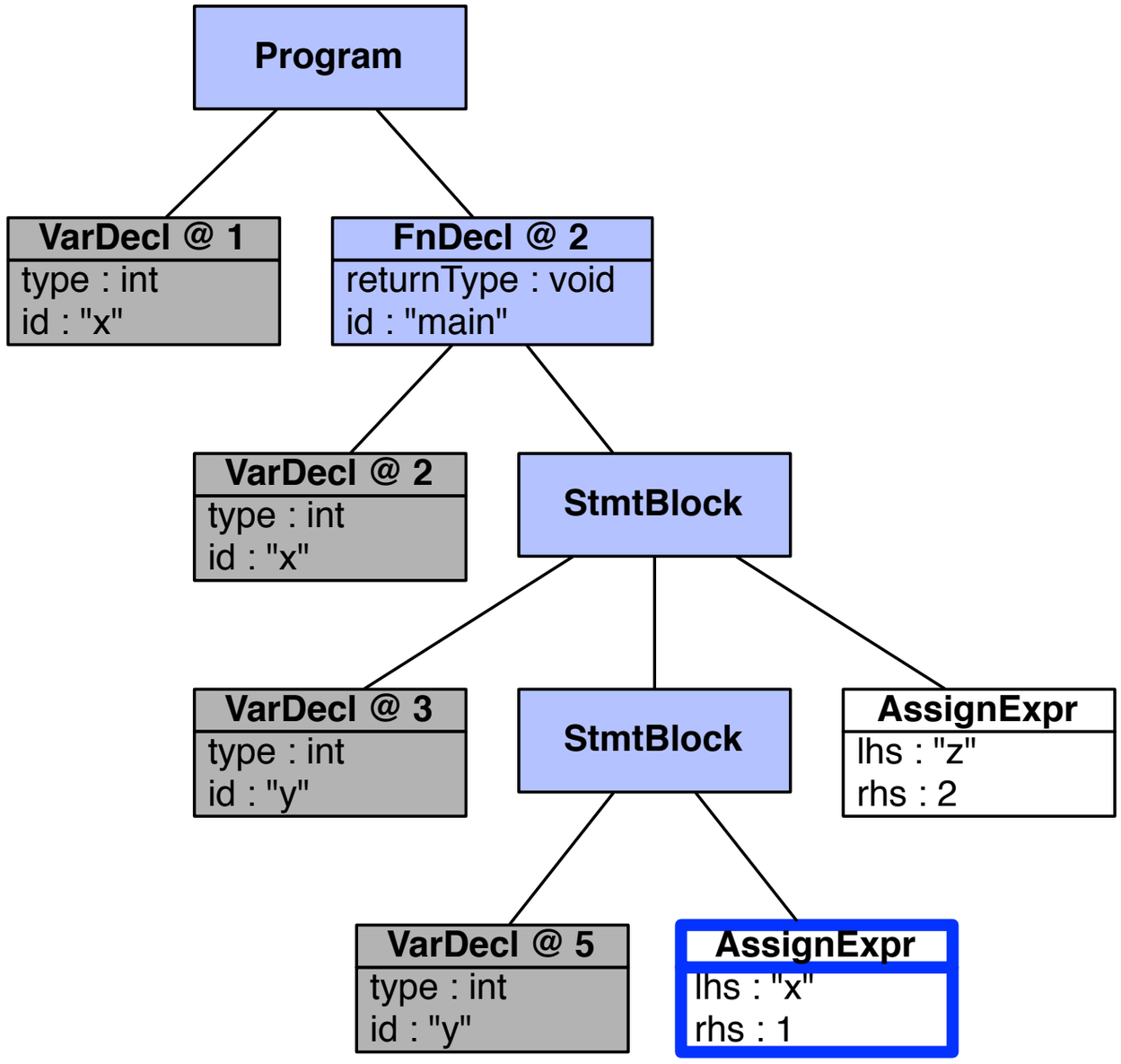


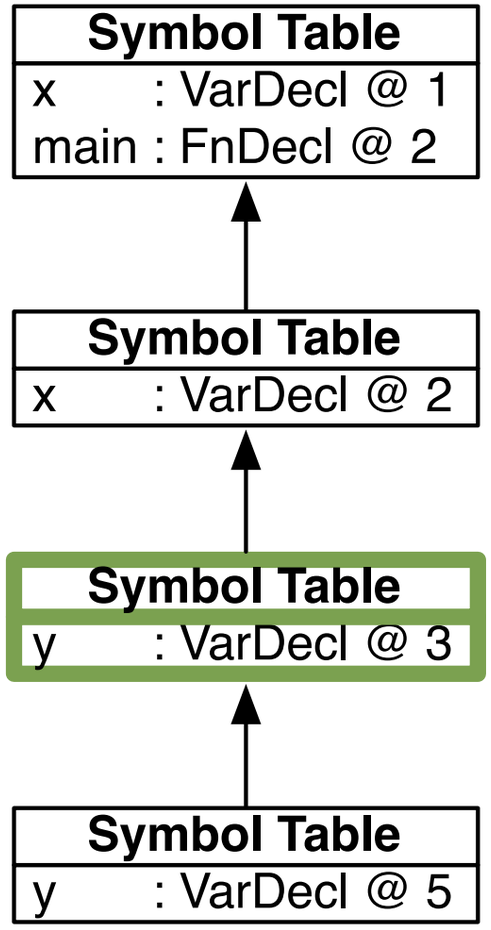
Scope System: Using Symbol Table



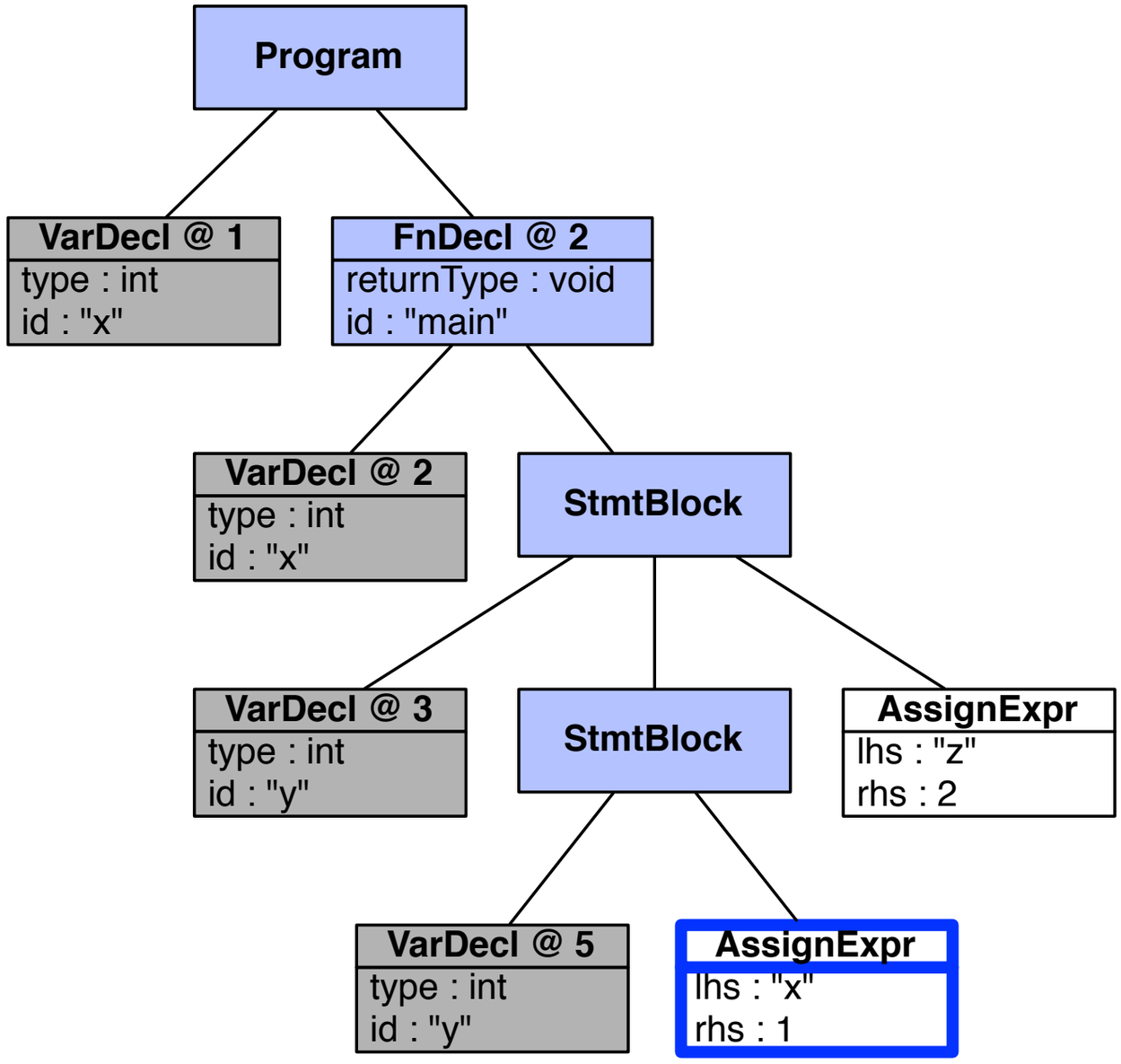


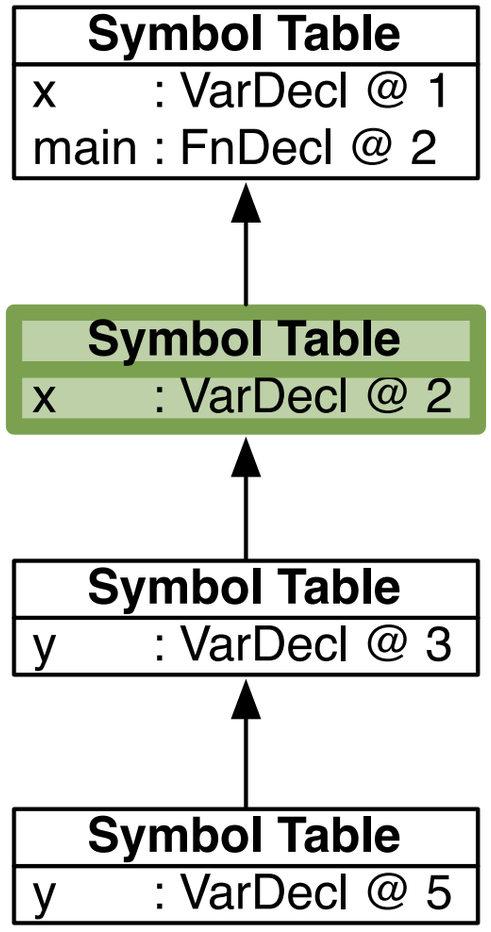
Scope System: Using Symbol Table



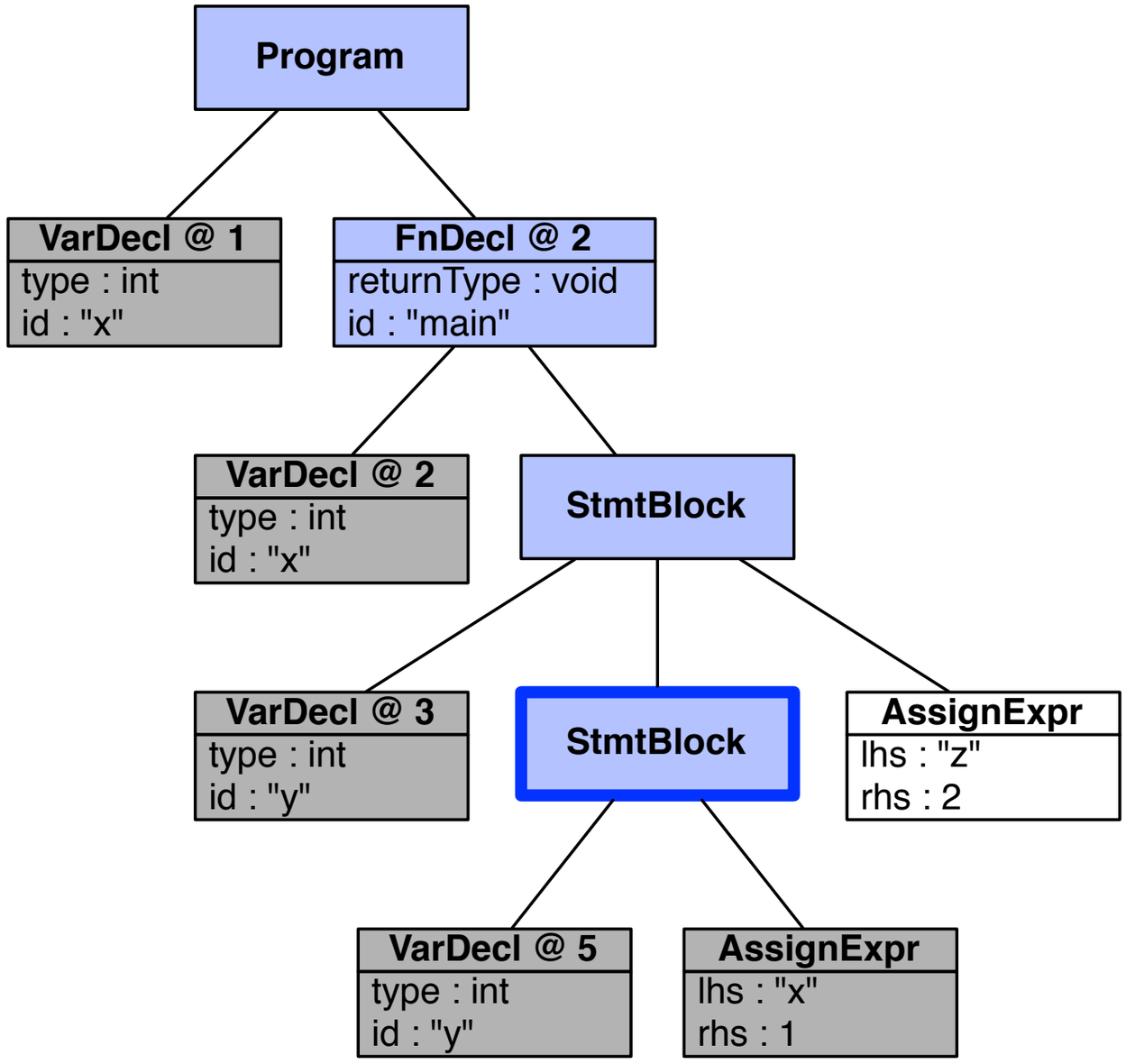


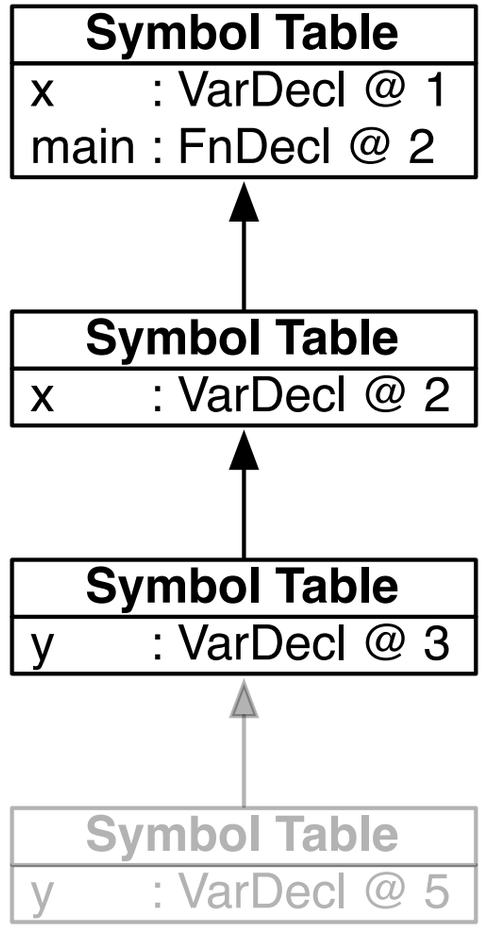
Scope System: Using Symbol Table



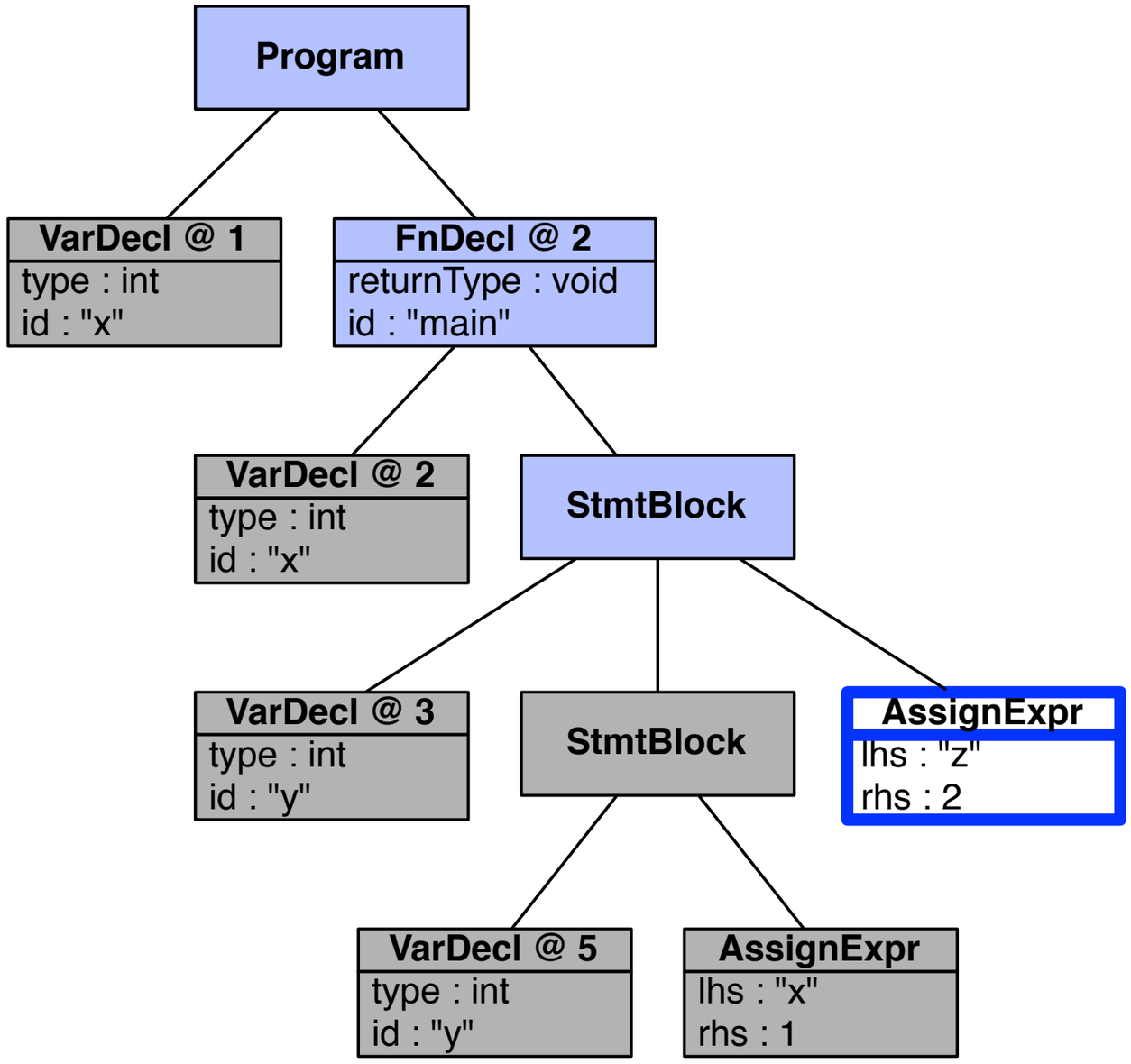


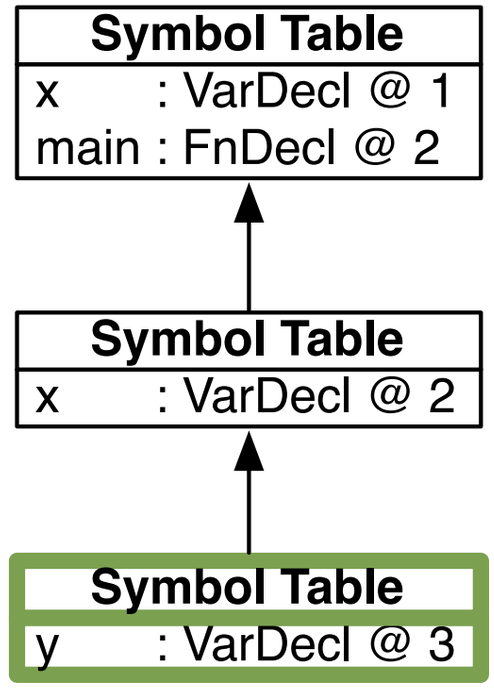
Scope System: Using Symbol Table



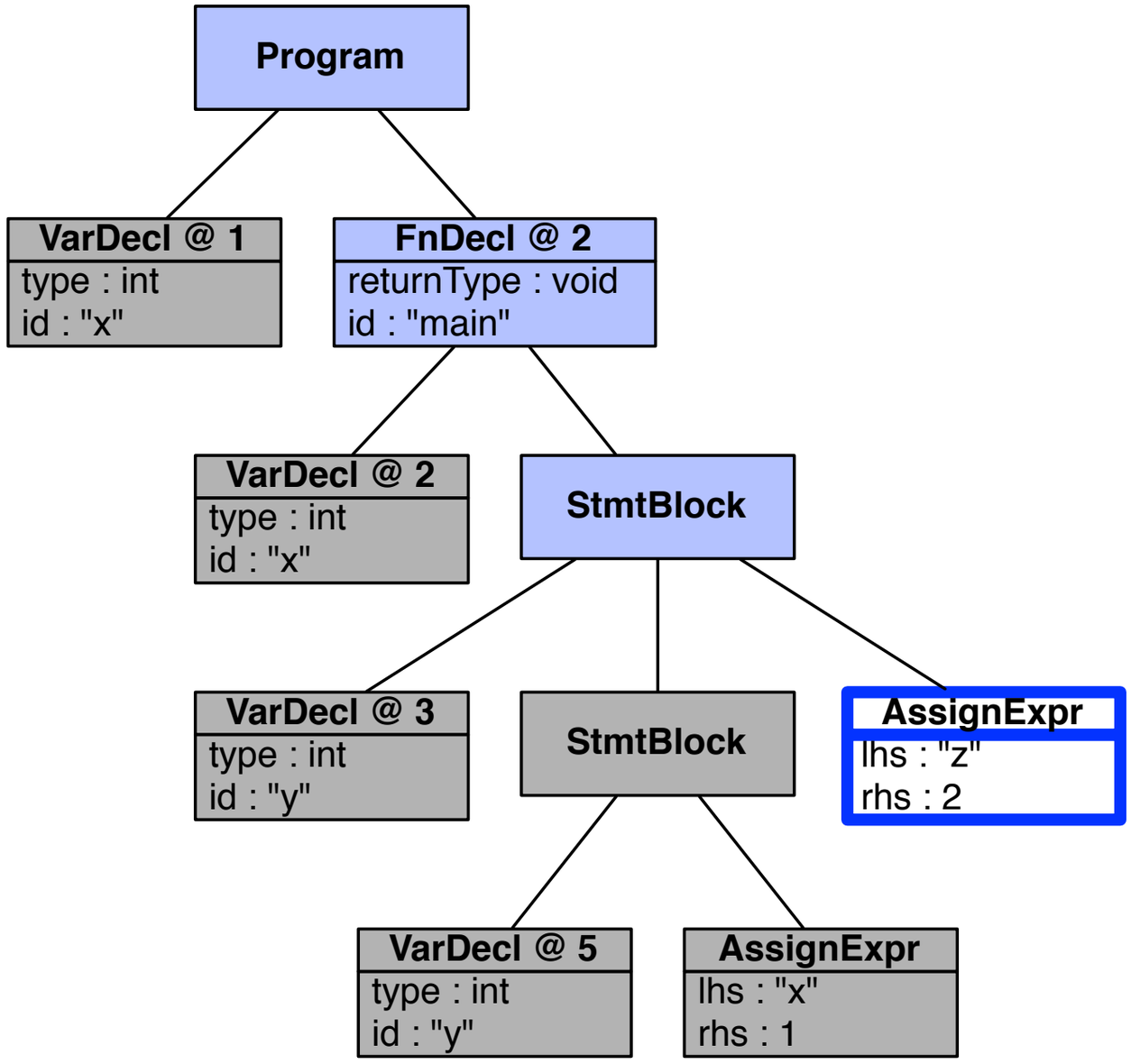


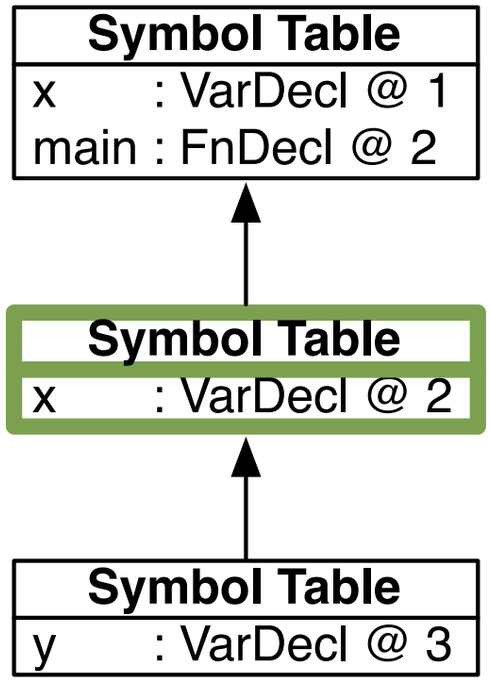
Scope System: Using Symbol Table



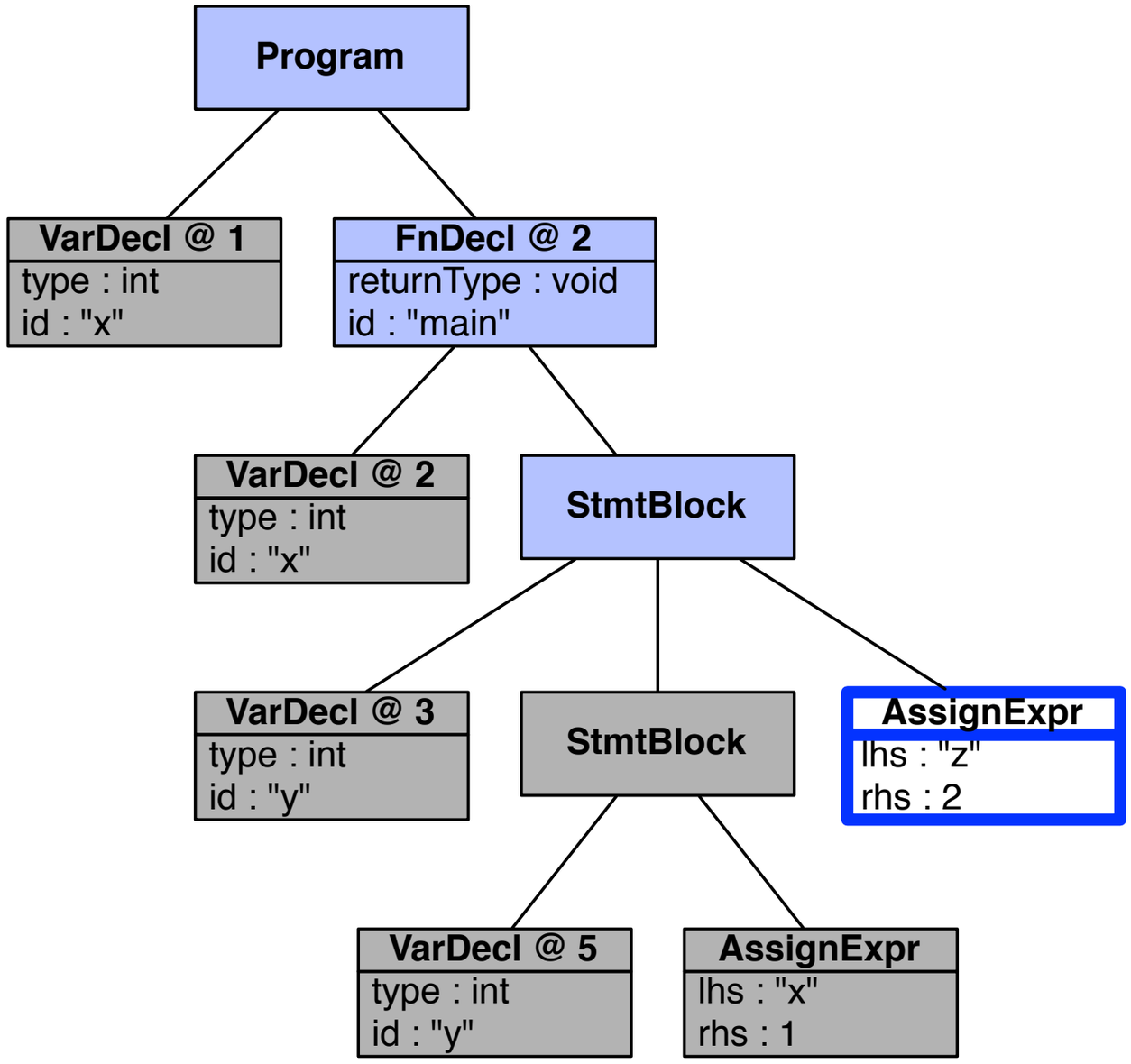


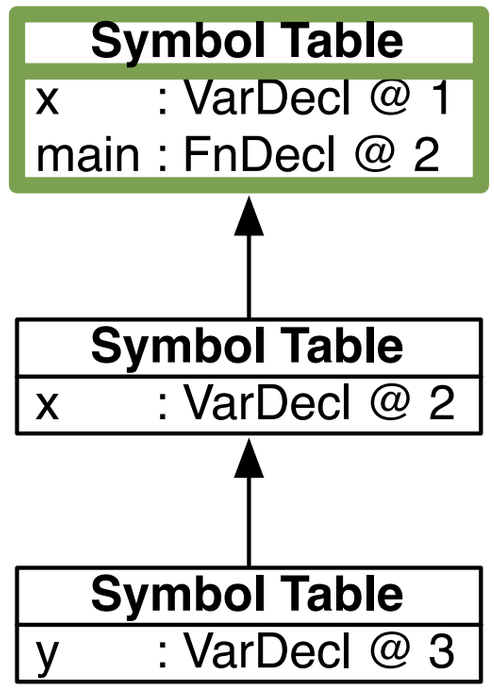
Scope System: Using Symbol Table





Scope System: Using Symbol Table

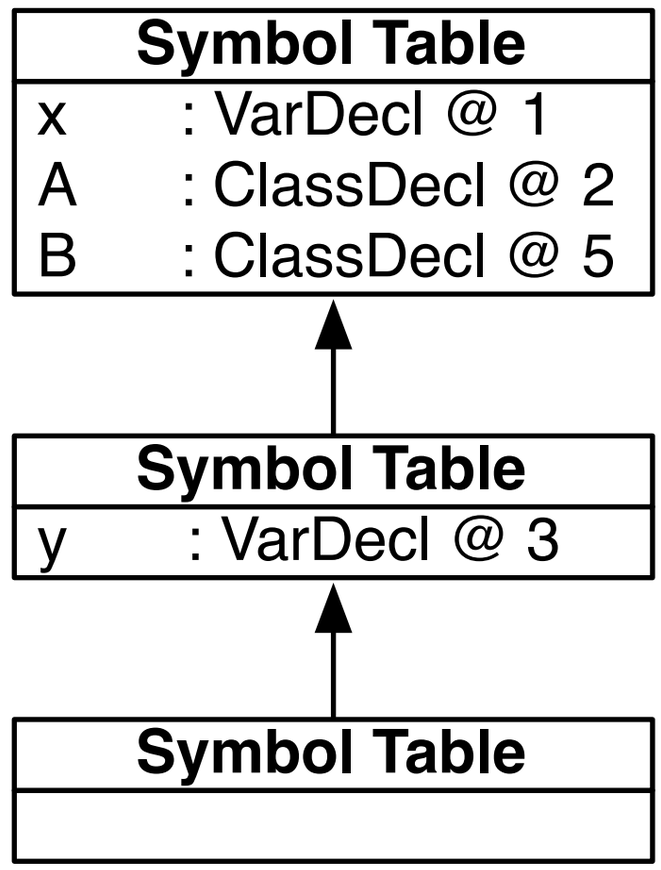




Error!

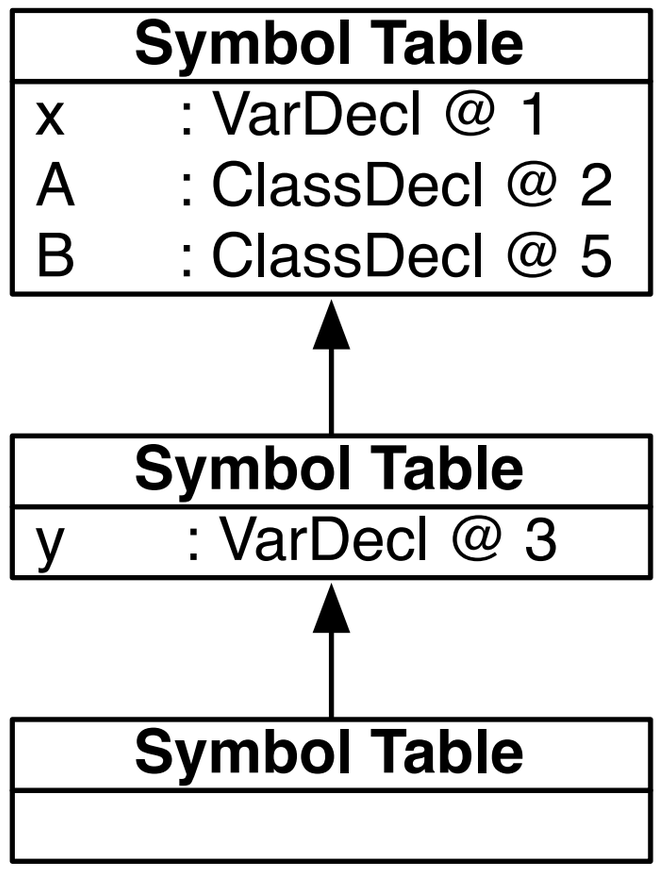
Class Scope

```
1 int x;
2 class A {
3       int y;
4 }
5 class B extends A {
6       void f() {
7             y = 5;
8       }
9 }
```



Class Declarations in Reverse Order

```
1 int x;
2 class B extends A {
3     void f() {
4         y = 5;
5     }
6 }
7 class A {
8     int y;
9 }
```



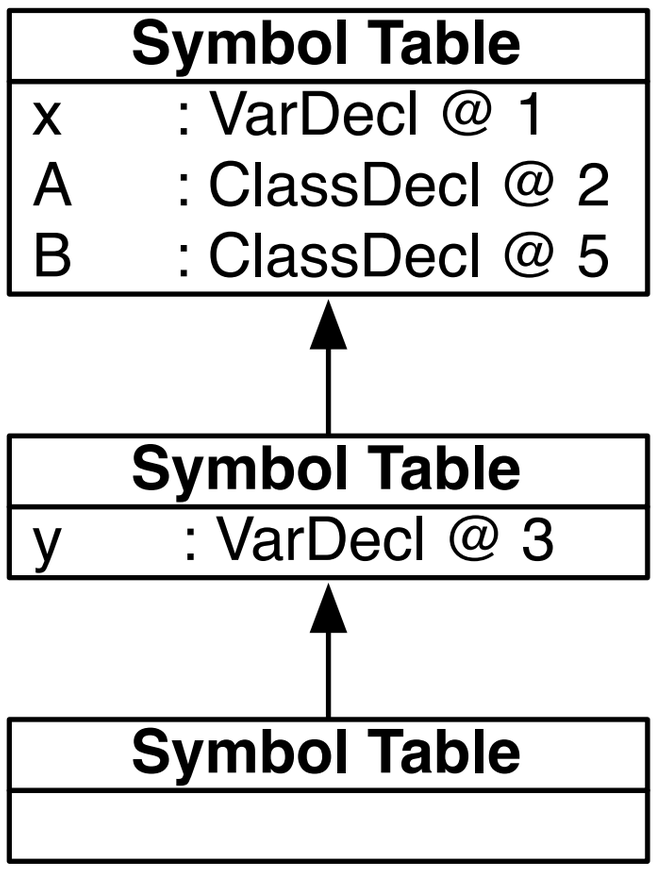
Go through all declarations in the scope first!

Reporting Errors in Lexical Order

```
1 int x;
2 class B extends A {
3     void f() {
4         z = 5;
5     }
6 }
7 class A {
8     unknown y;
9 }
```

Error!

Error!



Reporting Errors in Lexical Order

- First approach
  - Record but don't report an error when it is discovered
  - Sort all errors in lexical order and report them after the semantic analysis is finished
  - Trade space for time

- Second approach
  - When going through all declarations at the first time, don't report any error, just maintain the symbol tables
  - Go through the declarations again in lexical order and report the error
  - Trade time for space

Type System

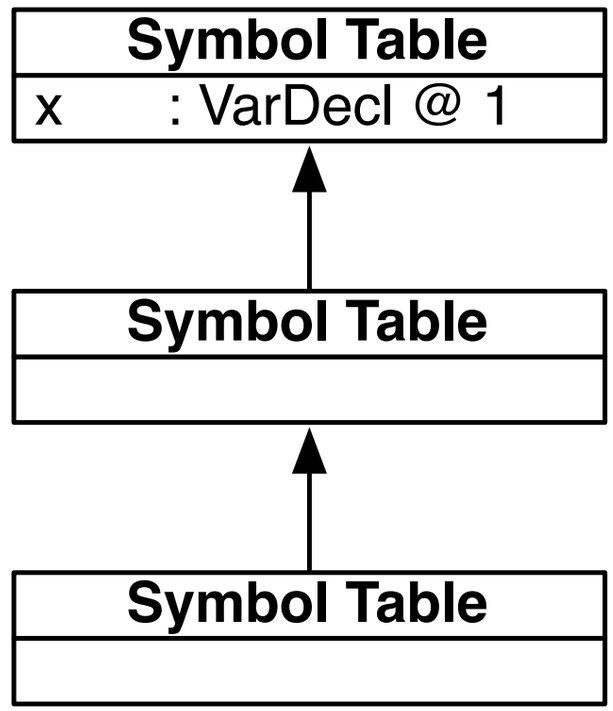
- Implement the typing rules in Node::Check()

$$\frac{S \vdash e_1 : T_1 \qquad S \vdash e_2 : T_2 \qquad T_2 \leq T_1}{S \vdash e1 = e2 : T_1}$$

```
AssignExpr::Check() {
    Type T1 = e1.GetType();
    Type T2 = e2.GetType();
    if (!T2.CompatibleWith(T1))
        ReportError;
    this->SetType(T1);
}
```

Dealing with Cascading Error

```
1 unknown x;          Error!
2 void main() {
3       x = 10;       Error!
4 }
```



- First approach
  - The second error won't exist once the first one is fixed
- Use ErrorType!

Thanks & Happy Vacation!