

Project 6 - Semantic Checking Part 2

Project 6: Semantic Checking (Part 2)

Due: Tuesday, November 29

Final: Monday, December 5, 10:15-12:05

Modify Checker.java

Catch all remaining errors

Fill in more fields in AST

PDF Files:

Assignment

List of all Error Messages

Project 6 - Semantic Checking Part 2

Need to insert “**implicit coercions**”

```
x := 1.2 + (i * 5);
```



```
x := 1.2 + intToReal(i * 5);
```

Project 6 - Semantic Checking Part 2

Need to insert “**implicit coercions**”

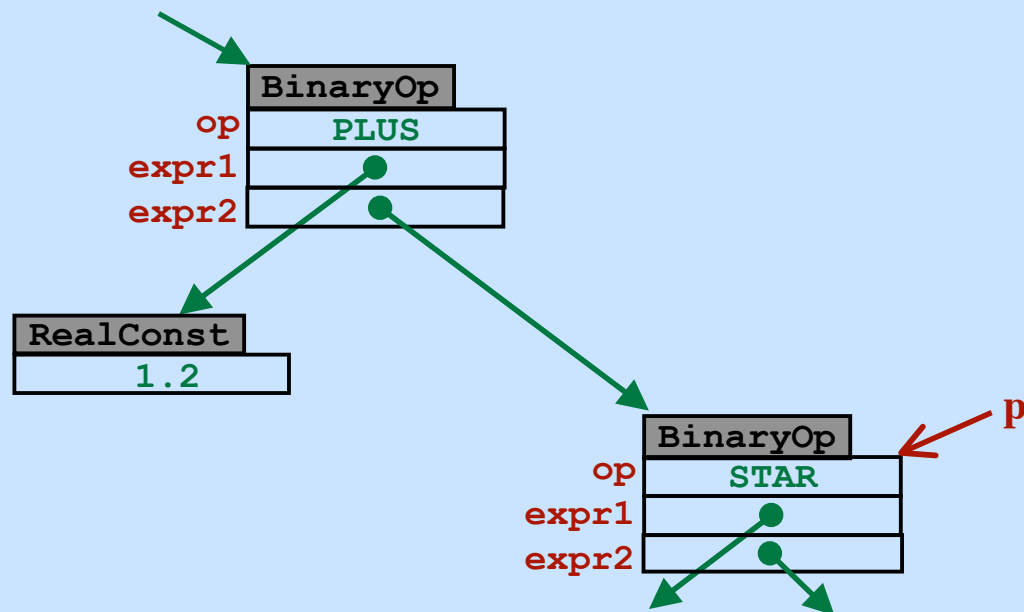
```
x := 1.2 + (i * 5);
```



```
x := 1.2 + intToReal(i * 5);
```

New Class: IntToReal

IntToReal
expr



Project 6 - Semantic Checking Part 2

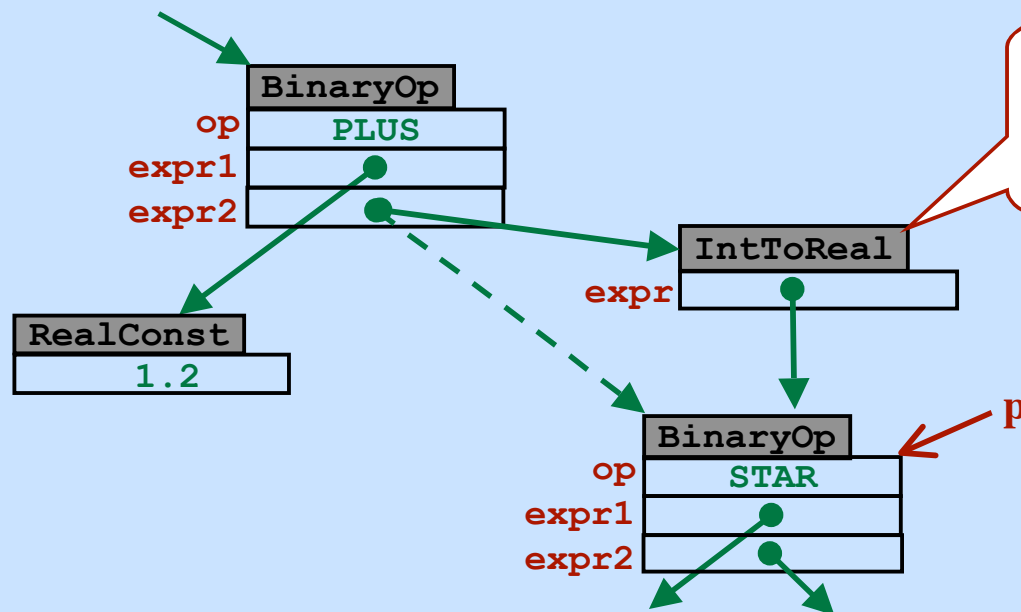
Need to insert “**implicit coercions**”

```
x := 1.2 + (i * 5);
```



```
x := 1.2 + intToReal(i * 5);
```

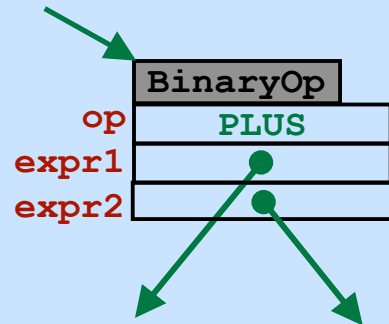
New Class: IntToReal



Project 6 - Semantic Checking Part 2

The “mode” field

What code to generate for:



?

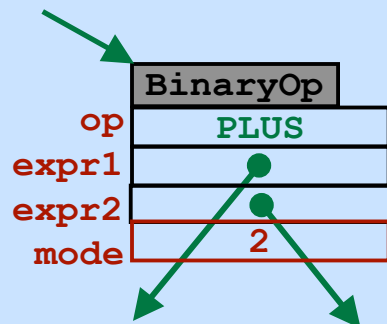
`iadd r2, r3, r5`

`fadd f3, f4, f5`

Project 6 - Semantic Checking Part 2

The “mode” field

What code to generate for:



? $\begin{cases} \text{iadd} & r2, r3, r5 \\ \text{fadd} & f3, f4, f5 \end{cases}$

The “mode” field added to...

BinaryOp
UnaryOp
ReadArg
Argument

Possible Values:

1 = INTEGER_MODE
2 = REAL_MODE
3 = STRING_MODE
4 = BOOLEAN_MODE

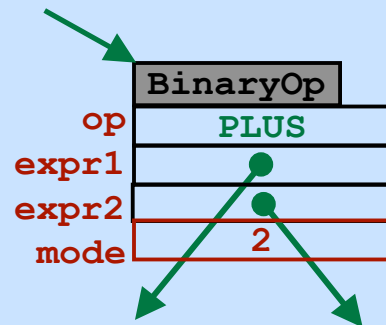
Add Java Constants...

```
static final int
  INTEGER_MODE = 1;
  REAL_MODE    = 2;
  STRING_MODE  = 3;
  BOOLEAN_MODE = 4;
```

Project 6 - Semantic Checking Part 2

The “mode” field

What code to generate for:



? $\begin{cases} \text{iadd} & r2, r3, r5 \\ \text{fadd} & f3, f4, f5 \end{cases}$

The “mode” field added to...

BinaryOp
UnaryOp
ReadArg
Argument

Possible Values:

1 = INTEGER_MODE
2 = REAL_MODE
3 = STRING_MODE
4 = BOOLEAN_MODE

Add Java Constants...

```
static final int
    INTEGER_MODE = 1;
    REAL_MODE    = 2;
    STRING_MODE  = 3;
    BOOLEAN_MODE = 4;
```

BOOLEAN_MODE and STRING_MODE will only be used for Arguments of WriteStmts.

```
write (a,b,c,d);
```

All values will be 32-bit binary values...

How shall we print each value?

Project 6 - Semantic Checking Part 2

Type Checking

Goal:

Check to make sure that the types are “correct”

$x := y;$

Need to check whether the type of x is equal to the type of y .

For the purposes of type checking...

we will need only the name of the type

TypeName

*Modify the “check” methods to return a **TypeName***

checkExpr
checkBinaryOp
...
checkValueOf
checkLValue

*Modify methods concerned with expressions
and L-Values to return the type of
the expression / L-Value.*

...
checkIfStmt
checkTypeDecl
...

Do not modify other methods

Project 6 - Semantic Checking Part 2

Type Equivalence

```
type T1 is record
    f: integer;
    g: real;
end;
T2 is record
    f: integer;
    g: real;
end;
```

```
var a: T1;
    b: T1;
    c: T2;
```

Is TypeOf (a) = TypeOf (b) ?
Is TypeOf (a) = TypeOf (c) ?

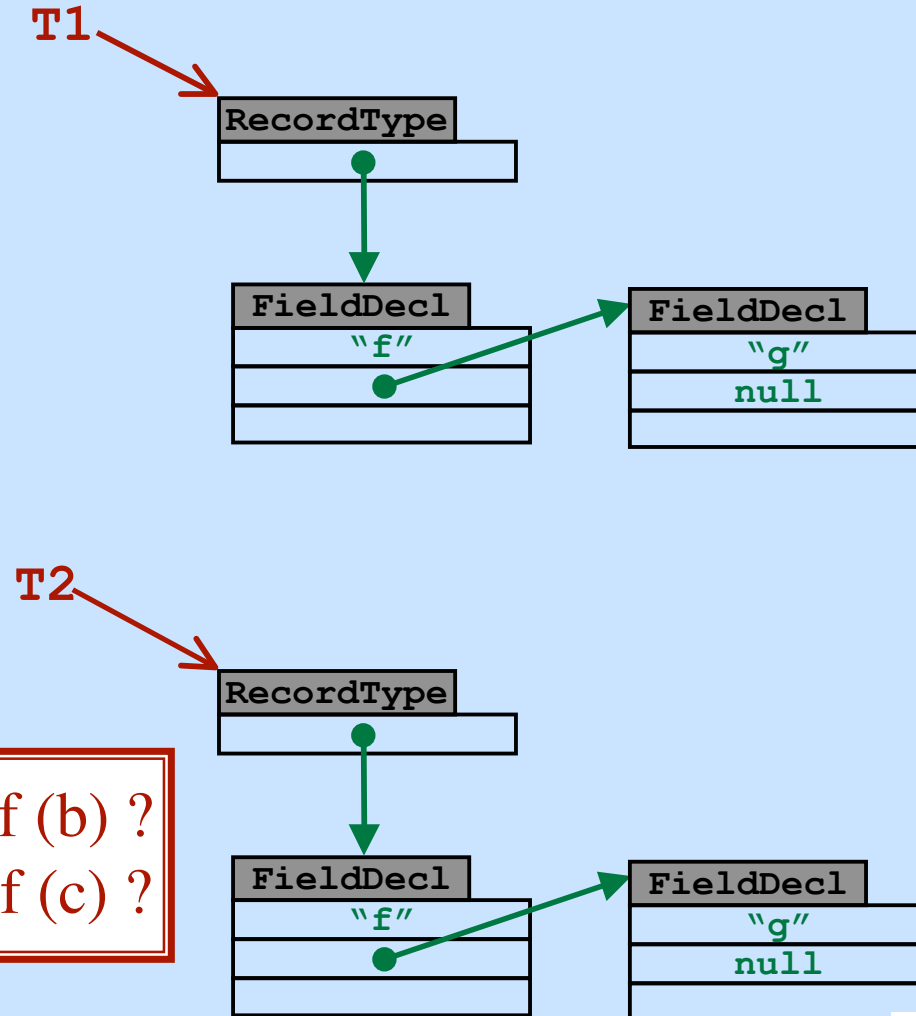
Project 6 - Semantic Checking Part 2

Type Equivalence

```
type T1 is record  
    f: integer; T1  
    g: real;  
end;  
T2 is record  
    f: integer;  
    g: real;  
end;
```

```
var a: T1;  
    b: T1;  
    c: T2;
```

Is TypeOf (a) = TypeOf (b) ?
Is TypeOf (a) = TypeOf (c) ?



Project 6 - Semantic Checking Part 2

Type Equivalence

```
type T1 is record  
    f: integer;  
    g: real;  
end;  
T2 is record  
    f: integer;  
    g: real;  
end;
```

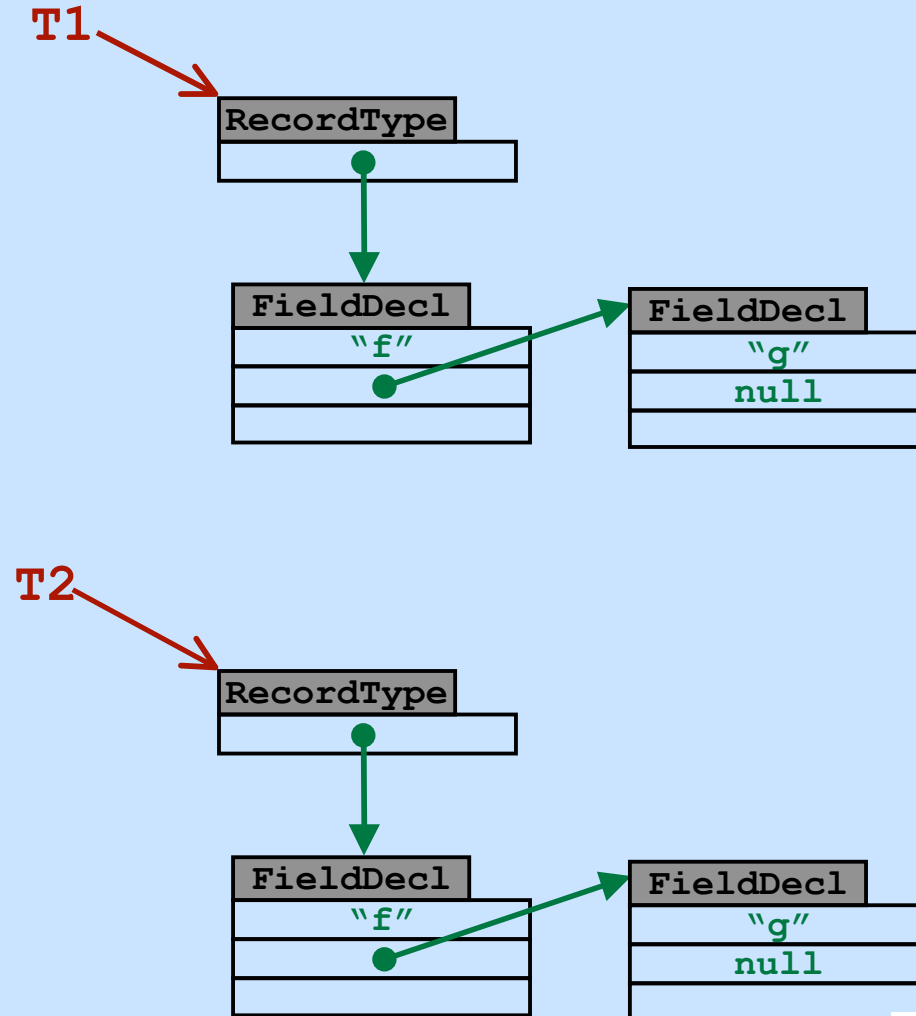
```
var a: T1;  
    b: T1;  
    c: T2;
```

Name Equivalence

TypeOf (a) = TypeOf (b)
TypeOf (a) ≠ TypeOf (c)

Structural Equivalence

TypeOf (a) = TypeOf (b)
TypeOf (a) = TypeOf (c)



Project 6 - Semantic Checking Part 2

Type Equivalence

```
type T1 is record  
    f: integer; T1  
    g: real;  
end;  
T2 is record  
    f: integer;  
    g: real;  
end;
```

```
var a: T1;  
    b: T1;  
    c: T2;
```

Name Equivalence

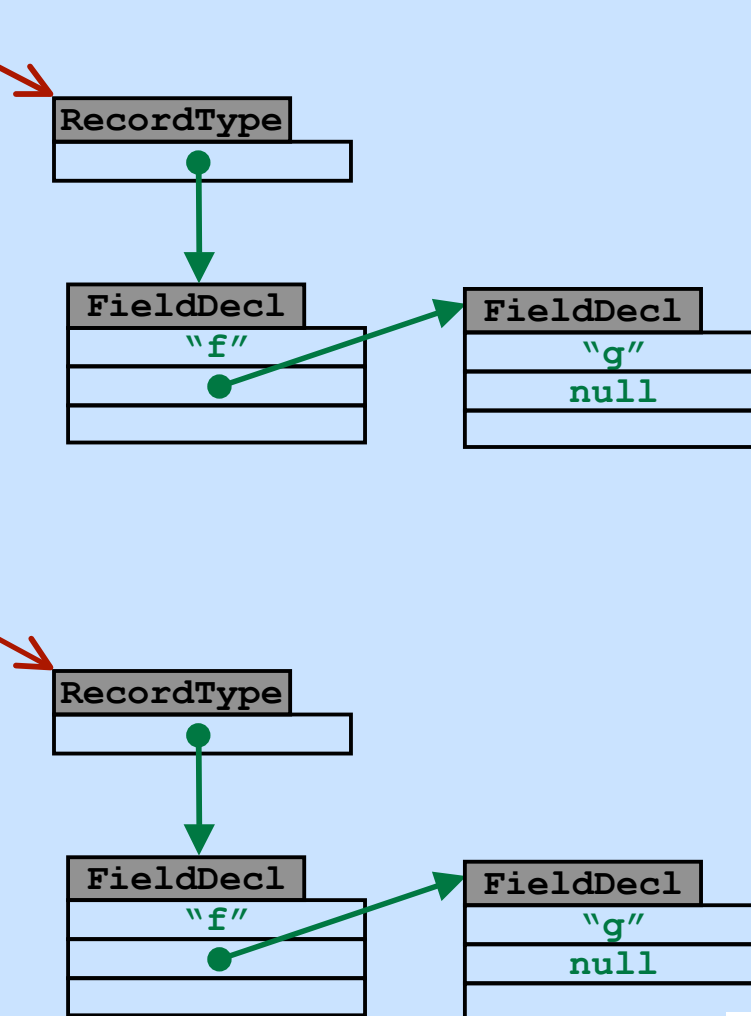
TypeOf (a) = TypeOf (b)
TypeOf (a) ≠ TypeOf (c)

Structural Equivalence

TypeOf (a) = TypeOf (b)
TypeOf (a) = TypeOf (c)

PCAT

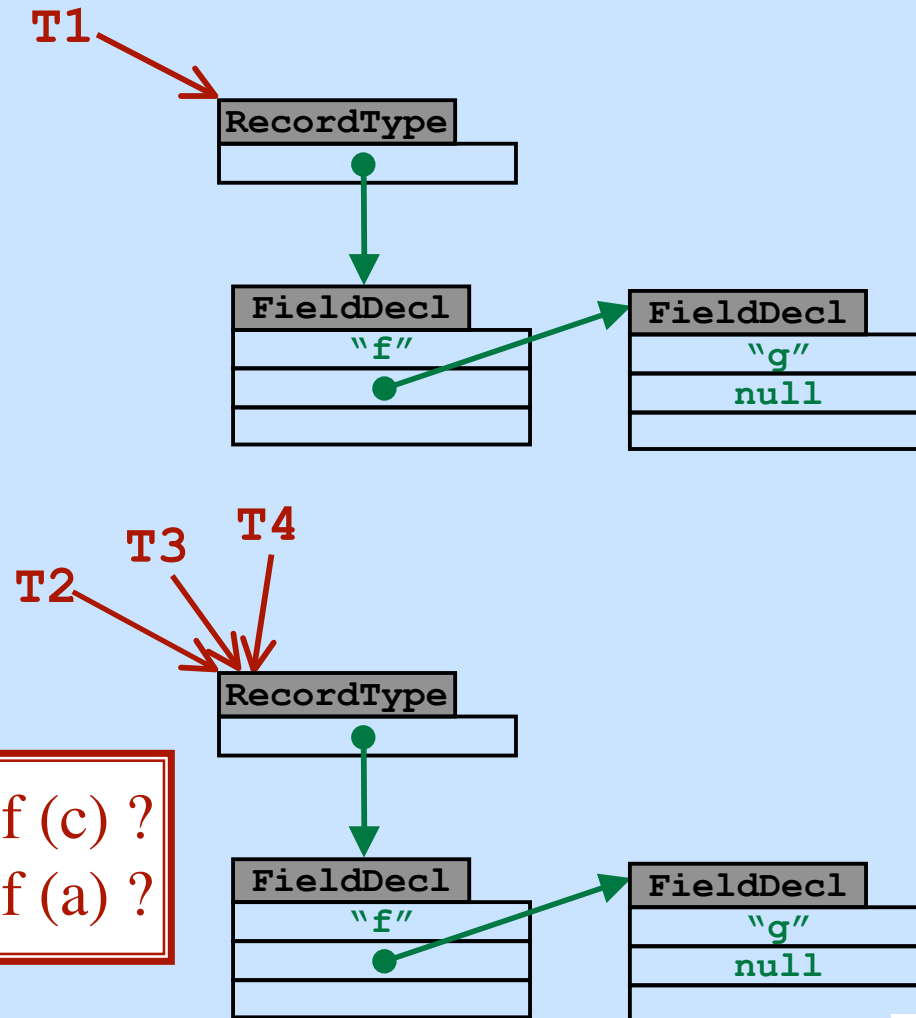
T2



Project 6 - Semantic Checking Part 2

Many Languages have "Type Aliasing"

```
type T1 is record  
    f: integer;  
    g: real;  
end;  
T2 is record  
    f: integer;  
    g: real;  
end;  
T3 is T2;  
T4 is T3;  
var a: T1;  
    b: T1;  
    c: T2;  
    d: T4;
```



Is $\text{TypeOf}(d) = \text{TypeOf}(c)$?
Is $\text{TypeOf}(d) = \text{TypeOf}(a)$?

Project 6 - Semantic Checking Part 2

Many Languages have "Type Aliasing"

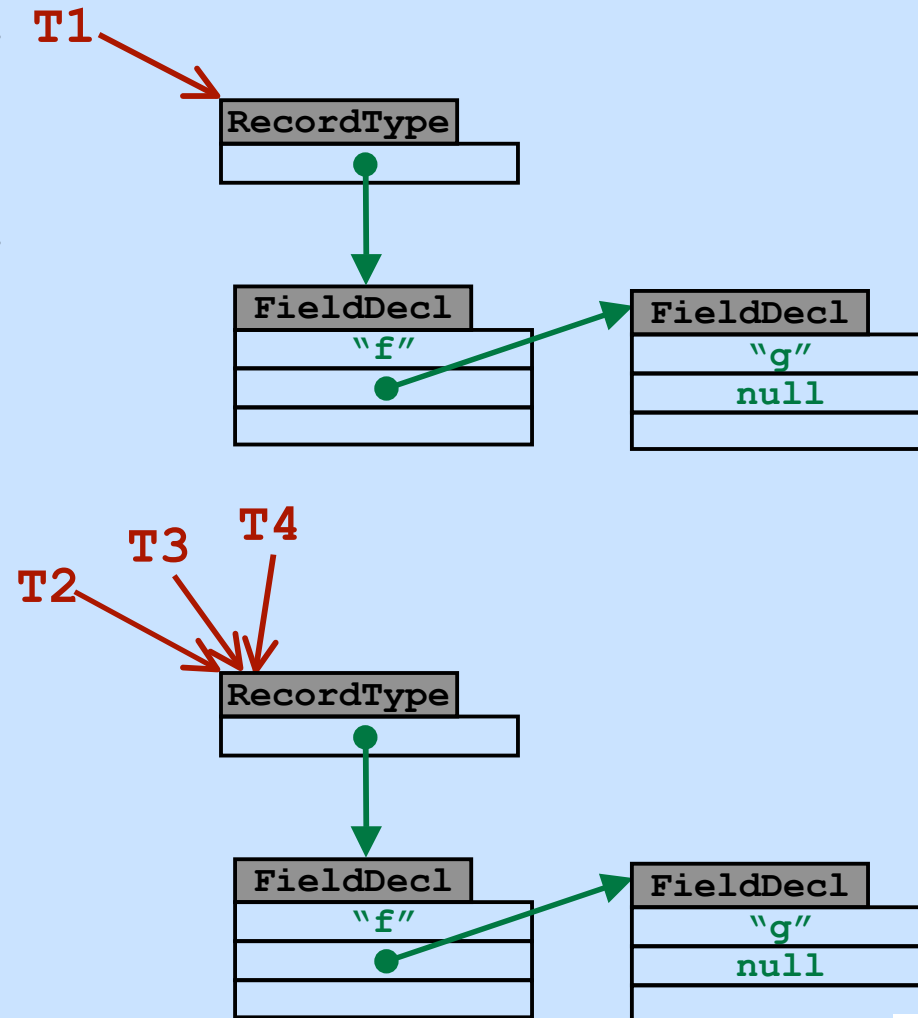
```
type T1 is record
  f: integer;
  g: real;
end;
T2 is record
  f: integer;
  g: real;
end;
T3 is T2;
T4 is T3;
var a: T1;
    b: T1;
    c: T2;
    d: T4;
```

Name Equivalence

TypeOf (d) = TypeOf (c)
TypeOf (d) \neq TypeOf (a)

Structural Equivalence

TypeOf (d) = TypeOf (c)
TypeOf (d) = TypeOf (a)

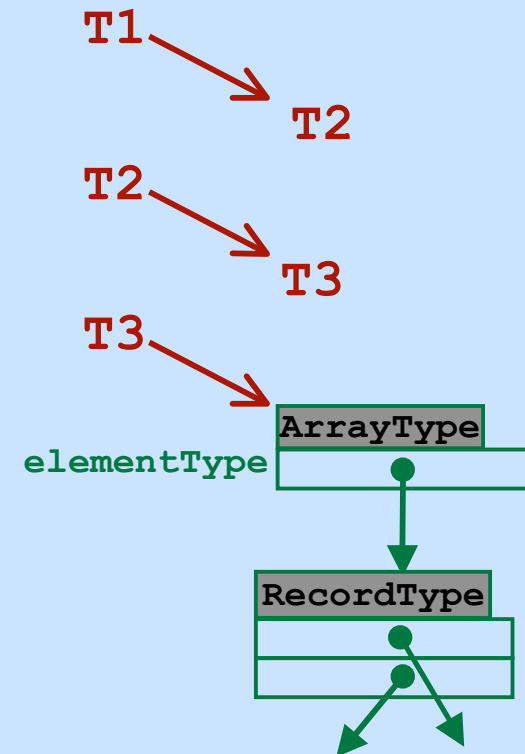


Project 6 - Semantic Checking Part 2

How to Deal with “Type Aliasing”

```
program is
  type T1 is T2;
        T2 is T3;
        T3 is array of record...;
  var x: T1 ...;
begin ... end;
```

What is the “real” type of x?



How to Deal with “Type Aliasing”

```
program is
  type T1 is T2;
      T2 is T3;
      T3 is array of record...;
  var x: T1 ...;
begin ... end;
```

What is the “real” type of x?

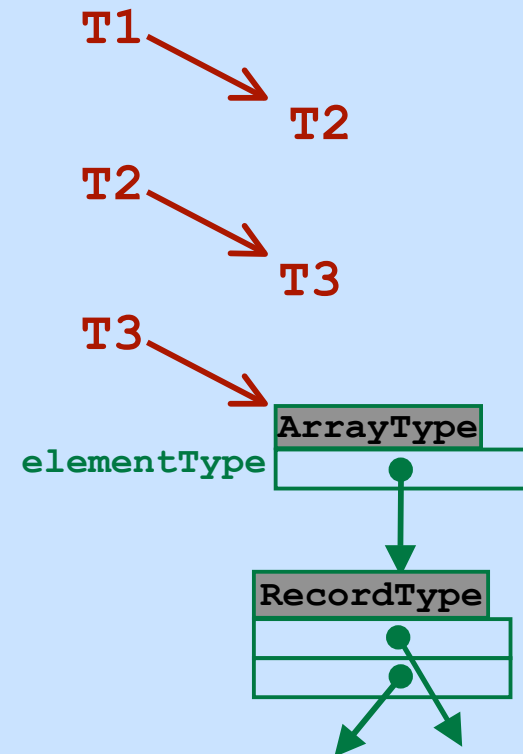
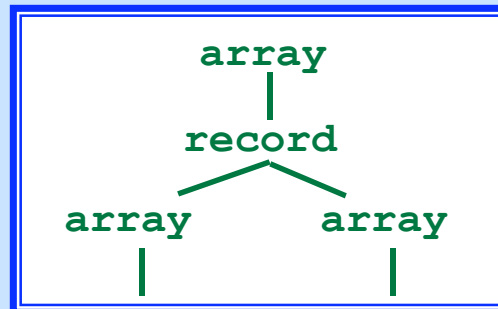
Note the distinction between

- Names of Types
- Underlying “concrete” Types

Types are represented by trees

array of record

```
f: array of int;
g: array of real;
end
```



Project 6 - Semantic Checking Part 2

How to Deal with “Type Aliasing”

```
program is
  type T1 is T2;
        T2 is T3;
        T3 is array of record...;
  var x: T1 ...;
begin ... end;
```

What is the “real” type of x?

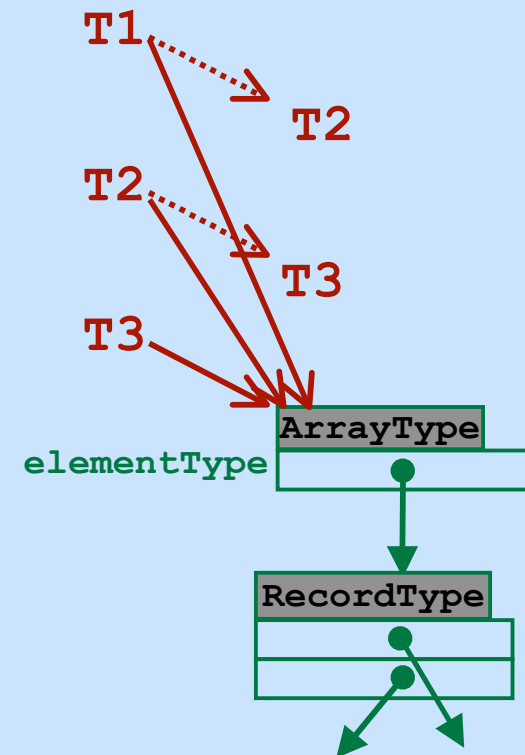
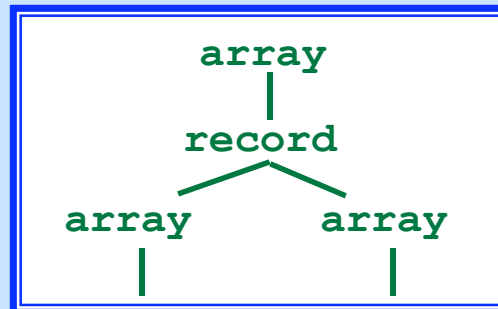
Note the distinction between

- Names of Types
- Underlying “concrete” Types

Types are represented by trees

array of record

```
f: array of int;
g: array of real;
end
```

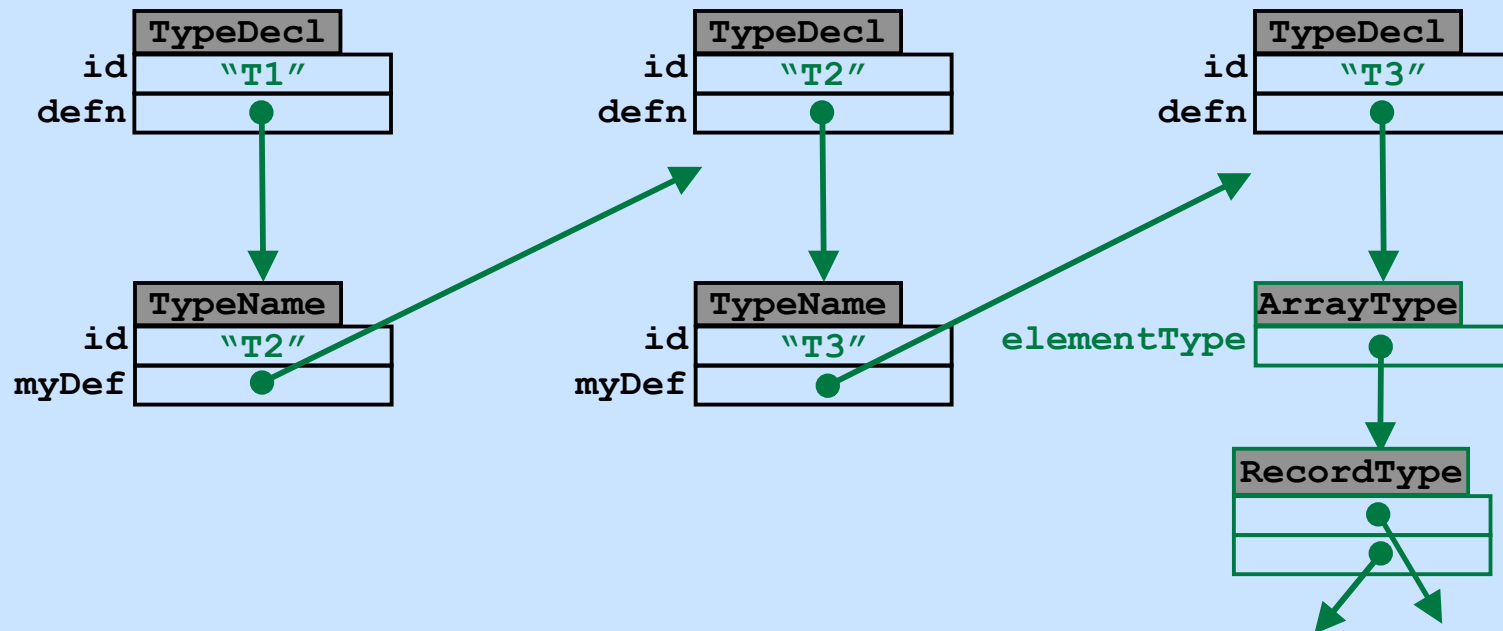


Project 6 - Semantic Checking Part 2

Concrete Types

```
program is
  type T1 is T2;
      T2 is T3;
      T3 is array of record...;
  var x: T1 ...;
  begin ... end;
```

What is the "real" type of x?

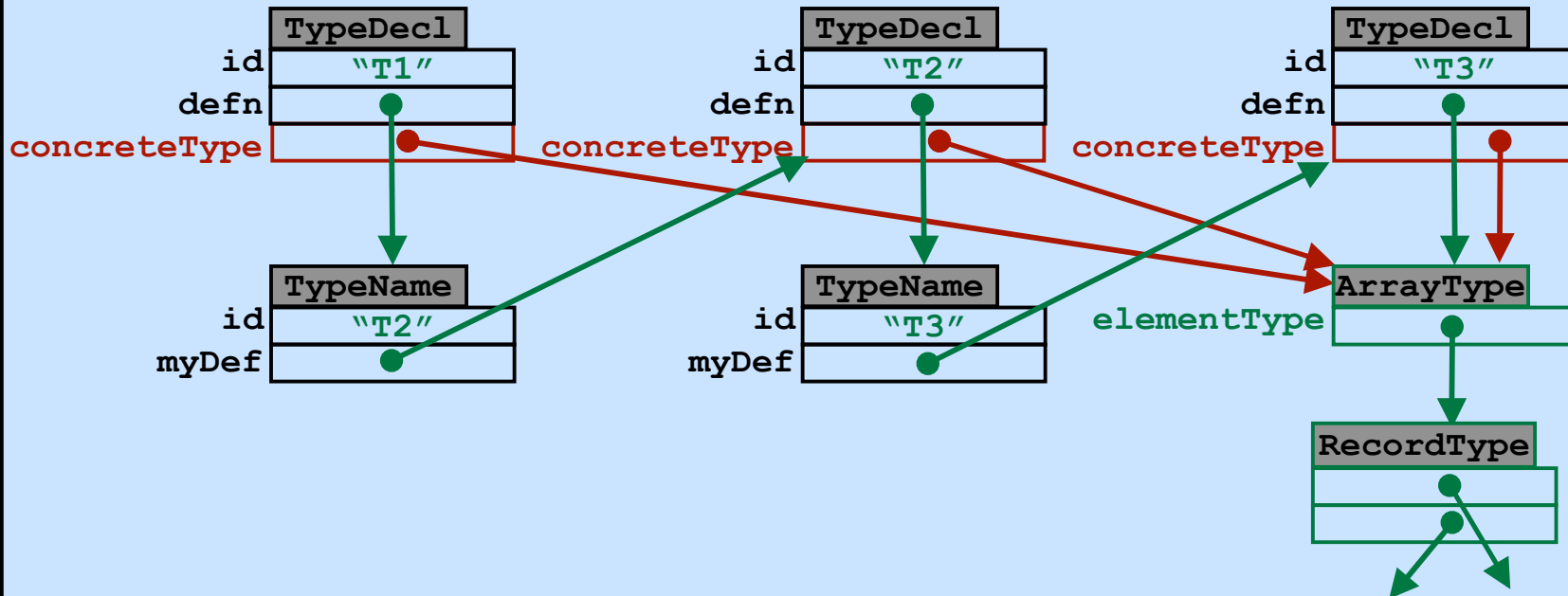


Project 6 - Semantic Checking Part 2

Concrete Types

```
program is
  type T1 is T2;
      T2 is T3;
      T3 is array of record...;
  var x: T1 ...;
  begin ... end;
```

What is the "real" type of x?

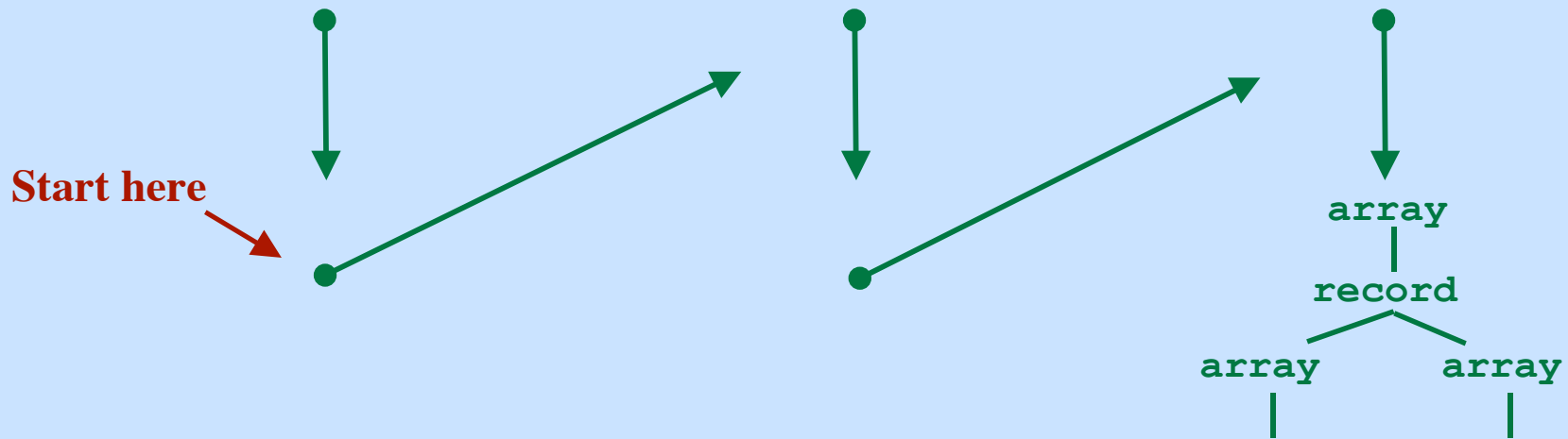


Project 6 - Semantic Checking Part 2

Finding the Underlying, Concrete Type

Goal:

Move through the definitions...
until we find a true, concrete type.
If we see another name, keep going.

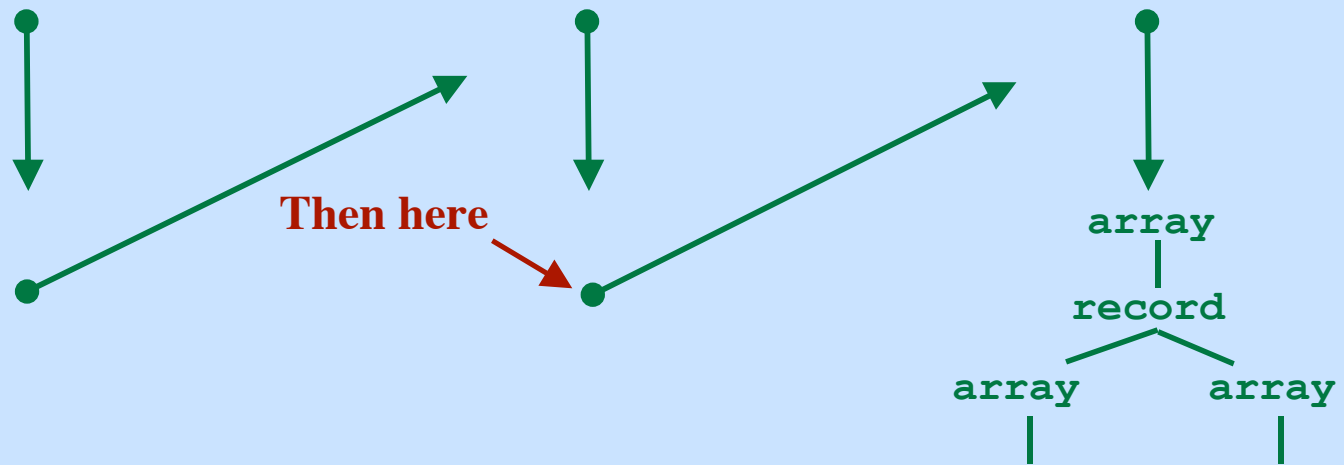


Project 6 - Semantic Checking Part 2

Finding the Underlying, Concrete Type

Goal:

Move through the definitions...
until we find a true, concrete type.
If we see another name, keep going.

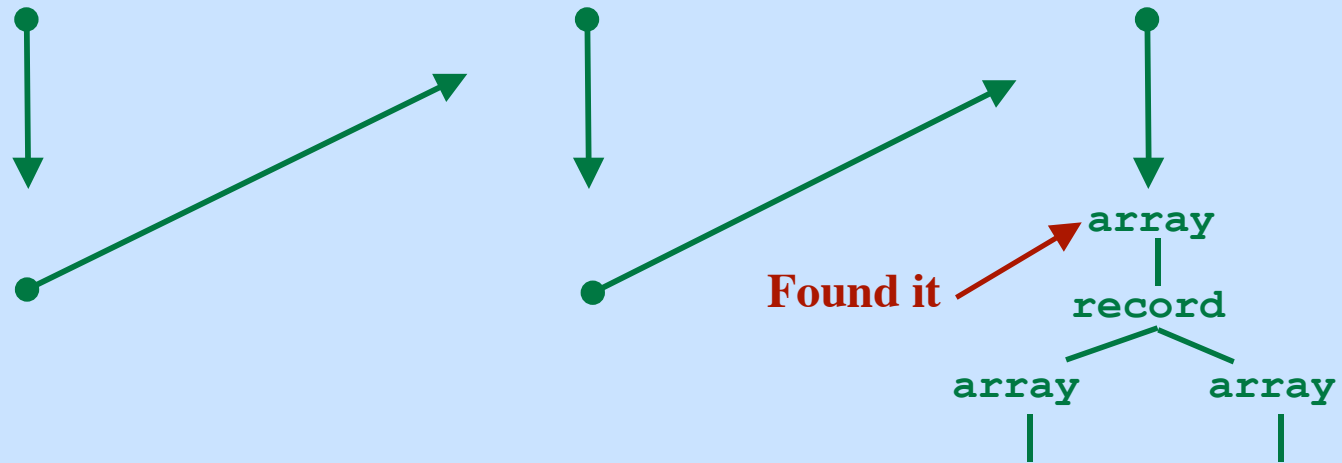


Project 6 - Semantic Checking Part 2

Finding the Underlying, Concrete Type

Goal:

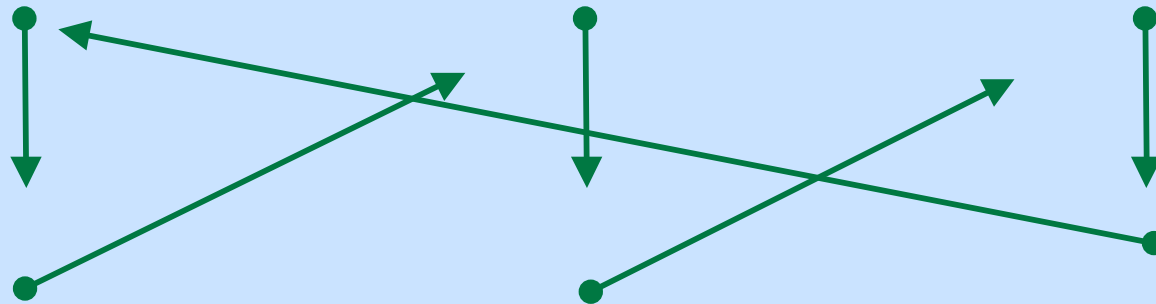
Move through the definitions...
until we find a true, concrete type.
If we see another name, keep going.



Project 6 - Semantic Checking Part 2

Problem: Cyclic Type Errors

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```



*We don't want to get caught in a cycle
... going around in circles forever!*

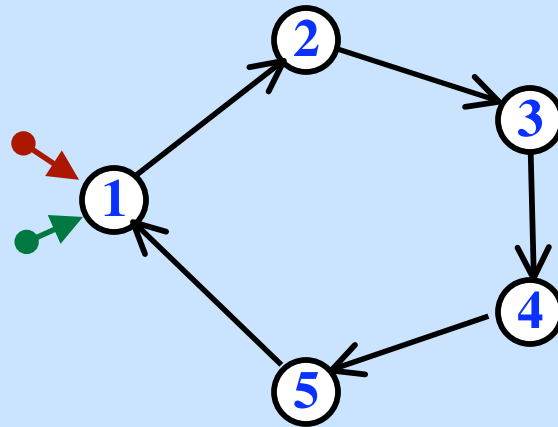
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



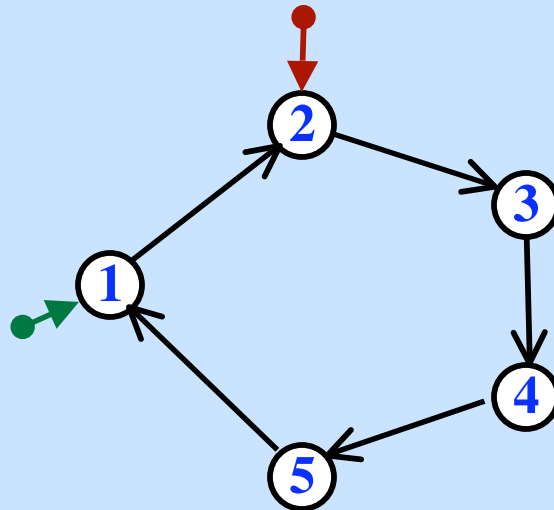
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



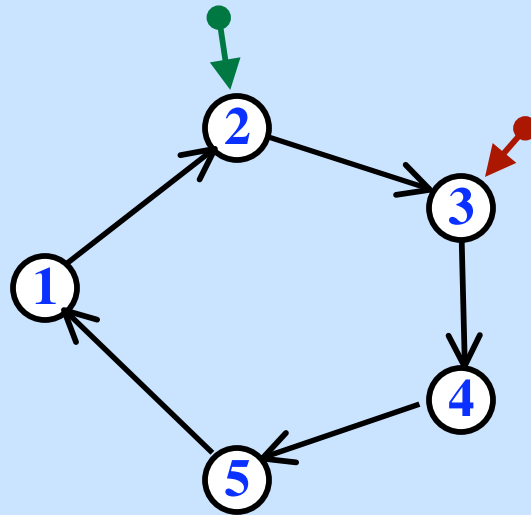
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



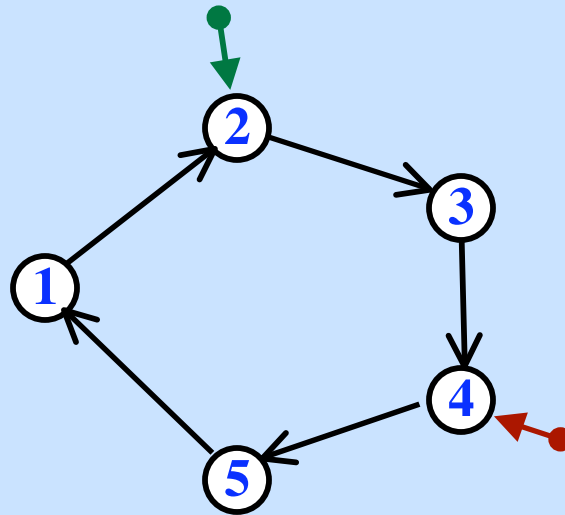
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



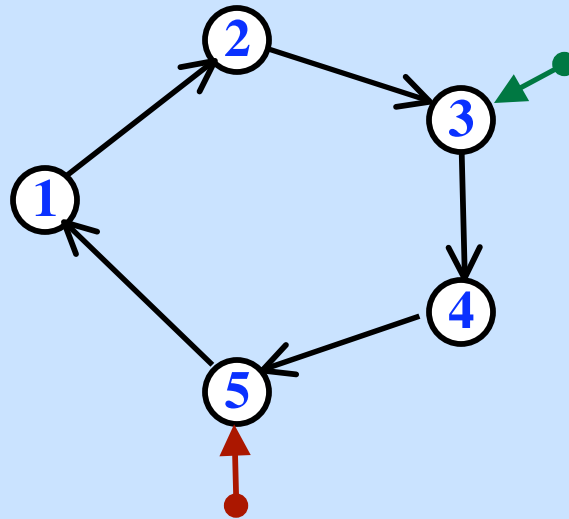
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



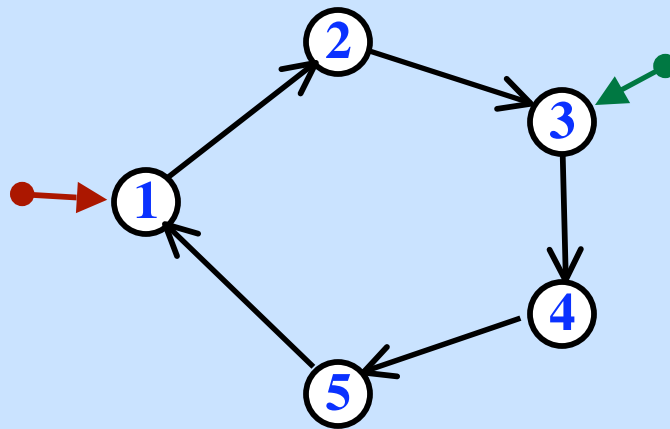
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



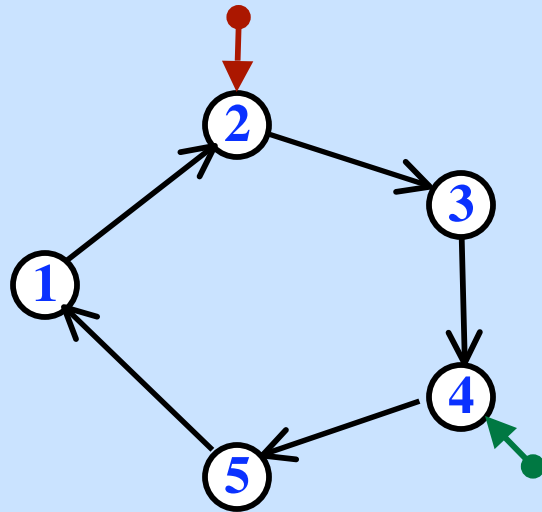
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



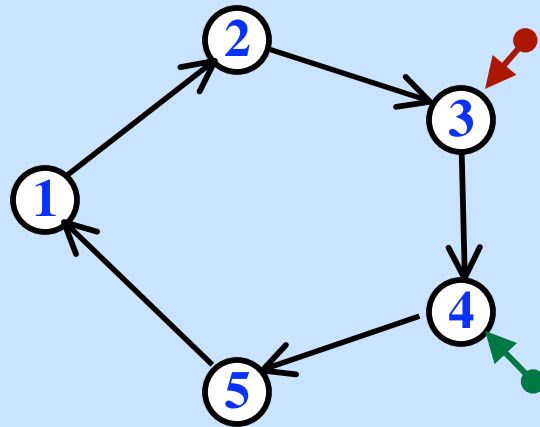
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



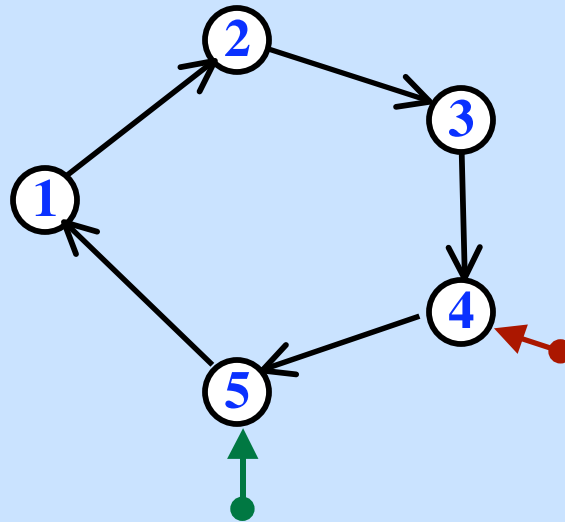
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



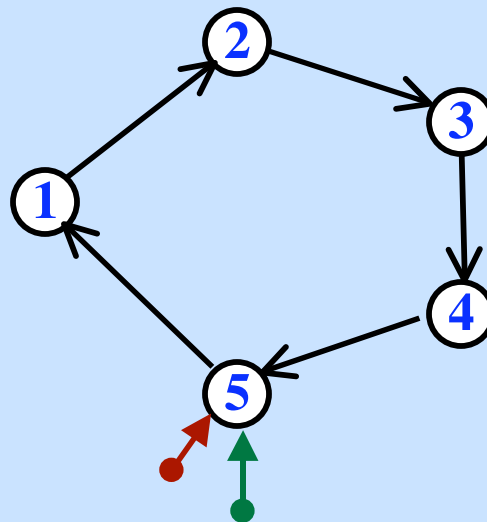
Project 6 - Semantic Checking Part 2

Cyclic Type Error

```
type T1 is T2;  
      T2 is T3;  
      T3 is T1;
```

Languages that allow type aliasing must detect this error.

Algorithm to find cycles in a graph:



*Pointer “p1” moves through the graph.
Pointer “p2” moves through the graph.
“p2” only moves every other time
Either:
“p1” reaches a concrete type
“p1” reaches “p2”
Finds the cycle in $2N$ steps.*

Checking Type Equality

Example:

```
program is  
  
  var x: integer;  
  procedure foo (...) is  
  
    var y: integer;  
    begin  
      ...  
      x := y;  
      ...  
    end;  
  begin  
    ...  
  end;
```

When the types are

integer

real

boolean

Then it is okay to

just compare the name IDs.

Checking Type Equality

Example:

```
program is
  type T1 is array of integer;
  var x: T1;
  procedure foo (...) is
    type T1 is array of boolean;
    var y: T1;
    begin
      ...
      x := y;
      ...
    end;
  begin
    ...
  end;
```

Is this assignment legal?
Must check whether
TypeOf(x) = TypeOf(y)
Can't just look at name of type!
When a type has a definition,
Must see if it is the same.

Project 6 - Semantic Checking Part 2

typeEquals

We often need to compare two types for equality.

Useful Routine: `typeEquals (Ast.TypeName t1, t2)` returns boolean

This method is passed two types .

Returns TRUE iff t1 and t2 are equal.

If *either* type name has a definition,

then we must compare definitions

“Name Equality”: Compare pointers (TypeName.myDef)

“Structural Equality”: Walk and compare the type trees

If both are undefined, then compare IDs.

Previous type errors during checking?

If either argument is NULL, just return TRUE

We want “Name Equality”

Example:

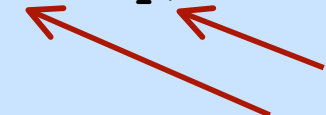
```
procedure foo (x: ...Error...) is begin ... end;  
...  
foo (7);
```

```
if !(typeEquals (__,__)) then  
    semanticError ("Type of argument is wrong");
```

Project 6 - Semantic Checking Part 2

Checking Assignment

x := y;



Check expr and get its type. Call it “fromType.”
Check l-value and get its type. Call is “toType.”

Useful Routine: `assignOK (Ast.Type to, from)` returns boolean

This method is passed two types .

Returns TRUE iff it is legal to assign from type “from” to type “to”.

When is this assignment legal?

$T_X = T_Y$ (Types are equal; use *typeEquals*)

$T_X = \text{Real}, T_Y = \text{Integer}$ (We'll also need a coercion)

$T_X = \text{NULL}$ or $T_Y = \text{NULL}$ (Due to previous errors)

$T_X = \text{ArrayType}$ and $T_Y = \text{NilType}$

$T_X = \text{RecordType}$ and $T_Y = \text{NilType}$

Need to use
`getCompoundType`
here

Project 6 - Semantic Checking Part 2

getCompoundType

Useful Routine: `getCompoundType (TypeName t) returns CompoundType`

This method is passed the name of a type. If it has a definition, then return a pointer to the ArrayType or RecordType.

`TypeName.myDef`

If the type has no definition, then return NULL

“integer”, “real”, “boolean”, or an undefined name

Errors? (the parameter may be NULL)

Return NULL

No error message

Example:

```
if (getCompoundType(t) instanceof ArrayType) ....
```

Adding the “intToReal” Coercion

Example:

```
r := i*j;
```

“fromType” ... integer?
“toType” ... real?

Useful Routine: `needCoercion (TypeName to, from) returns boolean`

This method is passed two types.

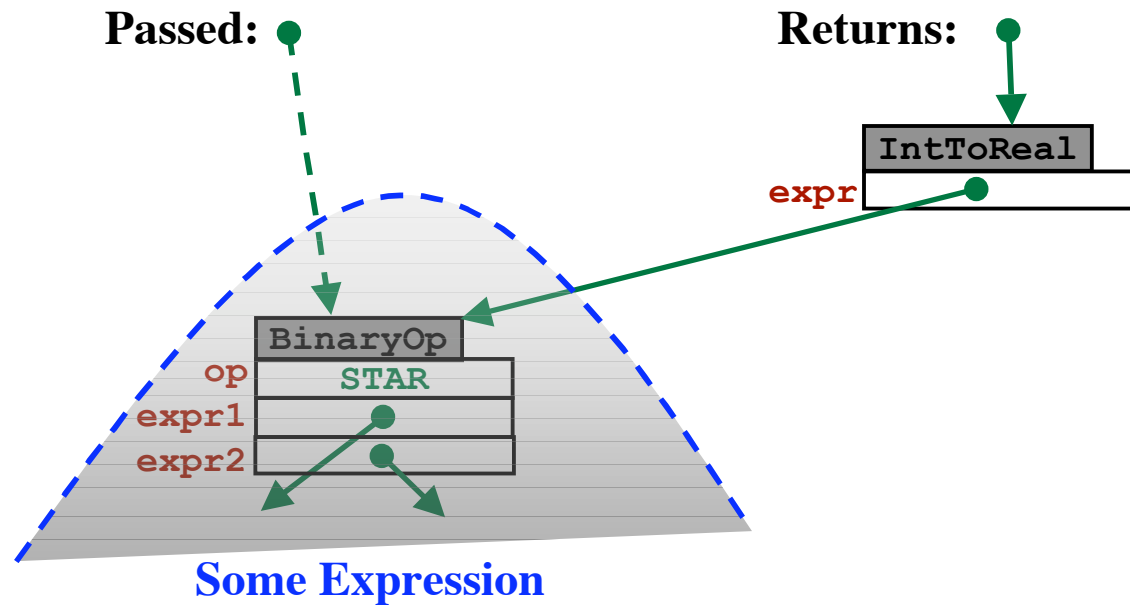
Returns TRUE iff it is legal to assign from type “from” to type “to”.

$T_{\text{FROM}}.\text{id} = \text{“integer”}$

$T_{\text{TO}}.\text{id} = \text{“real”}$

Adding the “intToReal” Coercion

Useful Routine: `insertCoercion (Ast.Expr t) returns Ast.IntToReal`



Project 6 - Semantic Checking Part 2

Example Code

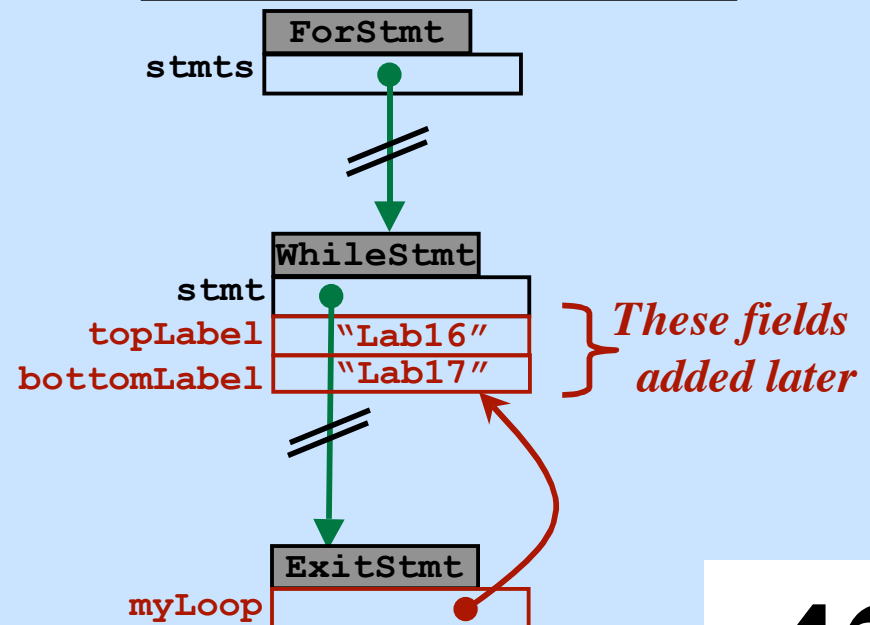
```
void checkAssignStmt (Ast.AssignStmt t) {  
  
    toType = checkLValue (t.lValue);  
    fromType = checkExpr (t.expr);  
  
    if (assignOK (toType, fromType)) then  
        if (needCoercion (toType, fromType)) then  
            t.expr = insertCoercion (t.expr);  
        endIf  
    else  
        semanticError (t.expr, "In assignment, type of  
            LHS is not compatible with type of RHS");  
    endIf  
  
}
```

Project 6 - Semantic Checking Part 2

New Field: myLoop

To generate code for EXIT statement
Will generate a branch to ... where?
Need to know which loop we are exiting from.

```
for ...  
...  
while condition do  
...  
exit;  
...  
end;  
...  
end;
```



Project 6 - Semantic Checking Part 2

New Field: myLoop

Later, we'll generate this code:

Lab16: } *Code for top of WHILE*

```
xxx  
xxx...goto Lab17...  
xxx
```

Code for condition

```
xxx  
xxx  
xxx
```

Code for statements

```
goto Lab17
```

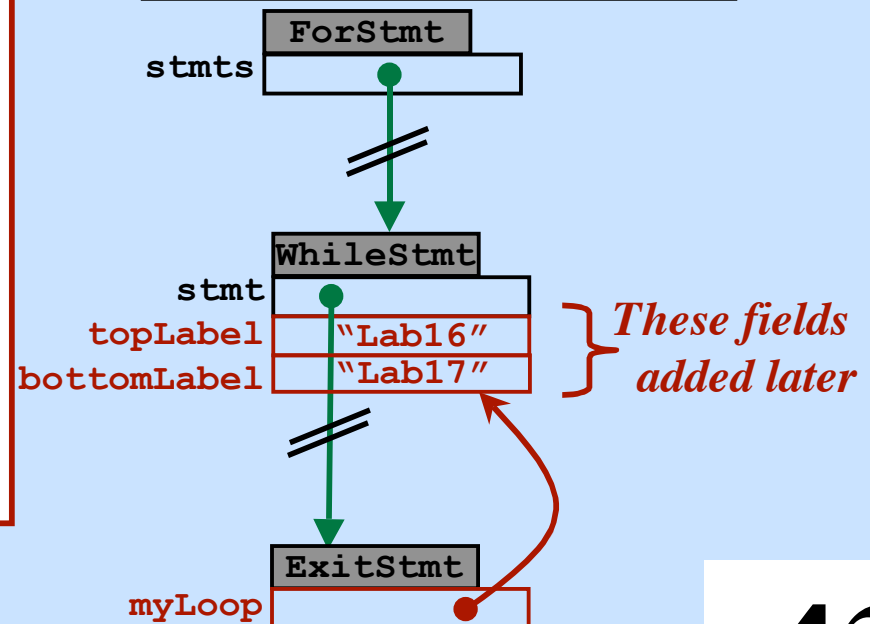
Code for EXIT

```
xxx  
xxx  
xxx
```

```
goto Lab16  
Lab17: }
```

Code for bottom of WHILE

```
for ...  
...  
while condition do  
...  
exit;  
...  
end;  
...  
end;
```



Project 6 - Semantic Checking Part 2

New Field: myProc

*To generate code for the RETURN statement,
need info about the procedure we are returning from .*

```
procedure foo (...): boolean is ...  
  ...  
  begin  
    ...  
    return [expression] ;  
    ...  
  end;
```

Is the expression required?

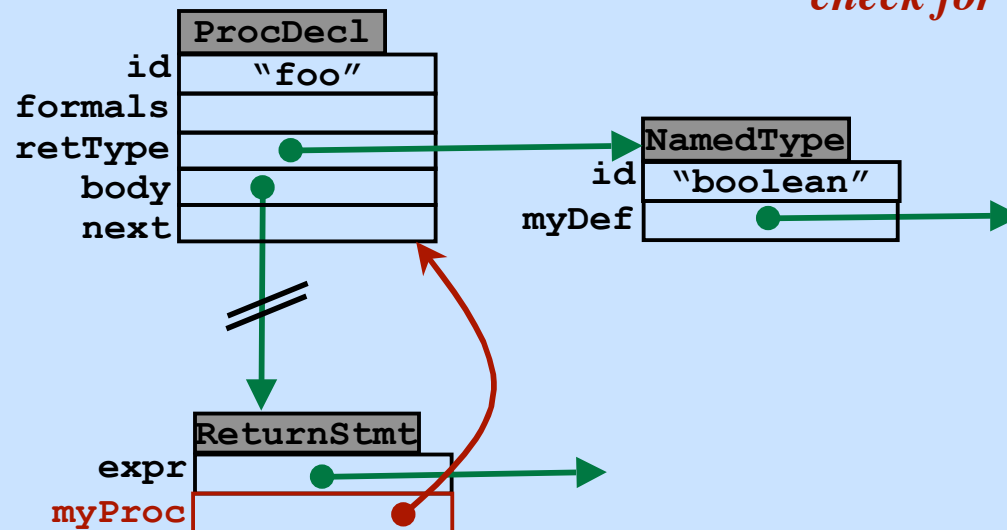
Must check!

Is the expression

*“assignment compatible”
with the return type?*

If the procedure is void,

check for $expr == NULL$.



Project 6 - Semantic Checking Part 2

Pass “currentLoop” and “currentProc” down through the check routines.

```
checkIfStmt (t, currentLoop, currentProc)
```

**Pointer to innermost LoopStmt,
WhileStmt, or ForStmt that
encloses this statement.
(NULL, if none)**

**Pointer to innermost ProcDecl
that encloses this statement.
(NULL, if none)**

```
checkReturnStmt (t, _, currentProc)
```

```
  t.myProc = currentProc;  
  if currentProc == NULL ... Error
```

```
checkExitStmt (t, currentLoop, _)
```

```
  t.myLoop = currentLoop;  
  if currentLoop == NULL ... Error
```

```
checkWhileStmt (t, _, currentProc)
```

```
  ...  
  checkStmts (t.stmts, t, currentProc)  
  ...
```

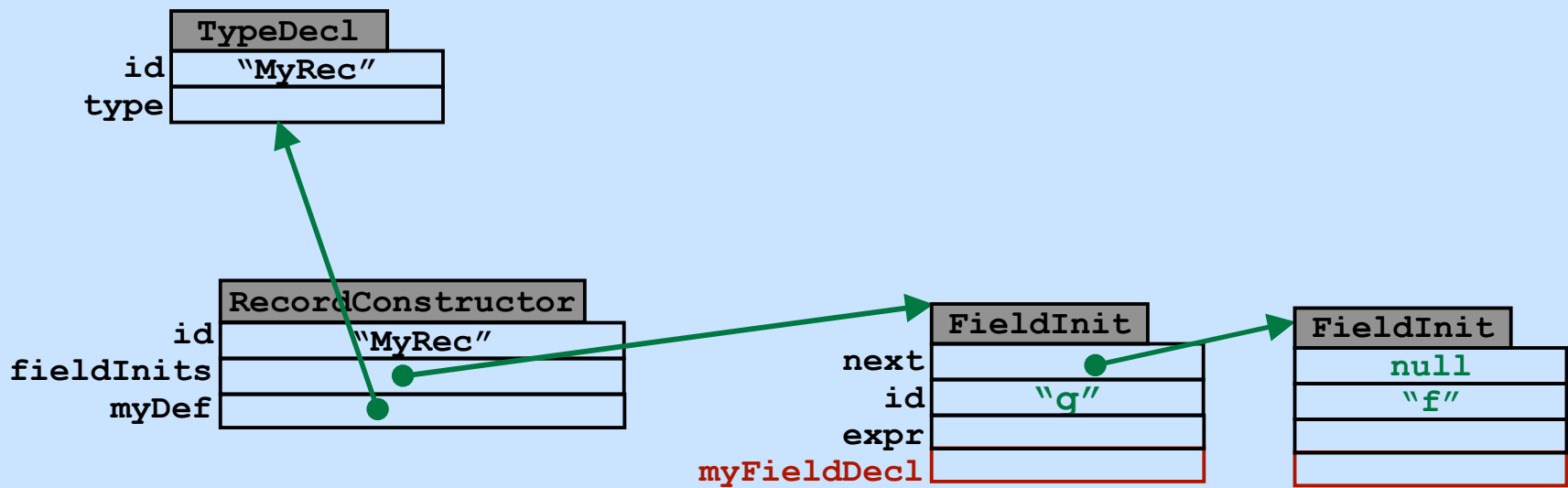
**Come methods will not
need currentLoop and/or
currentProc.**

**For those methods,
no need to pass it down.**

Project 6 - Semantic Checking Part 2

New Field "myFieldDecl"

```
type MyRec is record  
    f: integer;  
    g: real;  
    end;  
var r: MyRec := nil;  
...  
r := MyRec { g := 1.2; f := 5 };
```

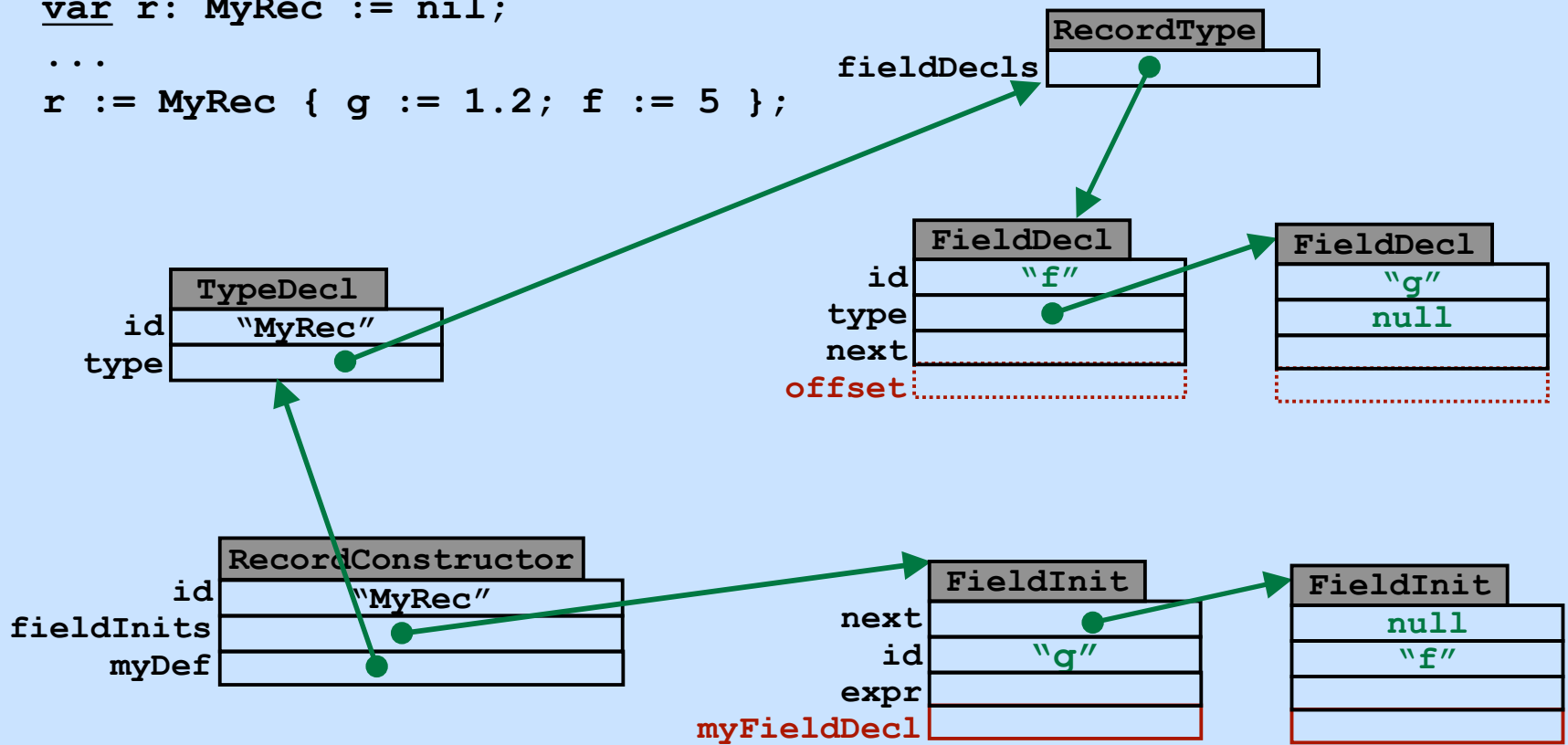


Project 6 - Semantic Checking Part 2

New Field "myFieldDecl"

```

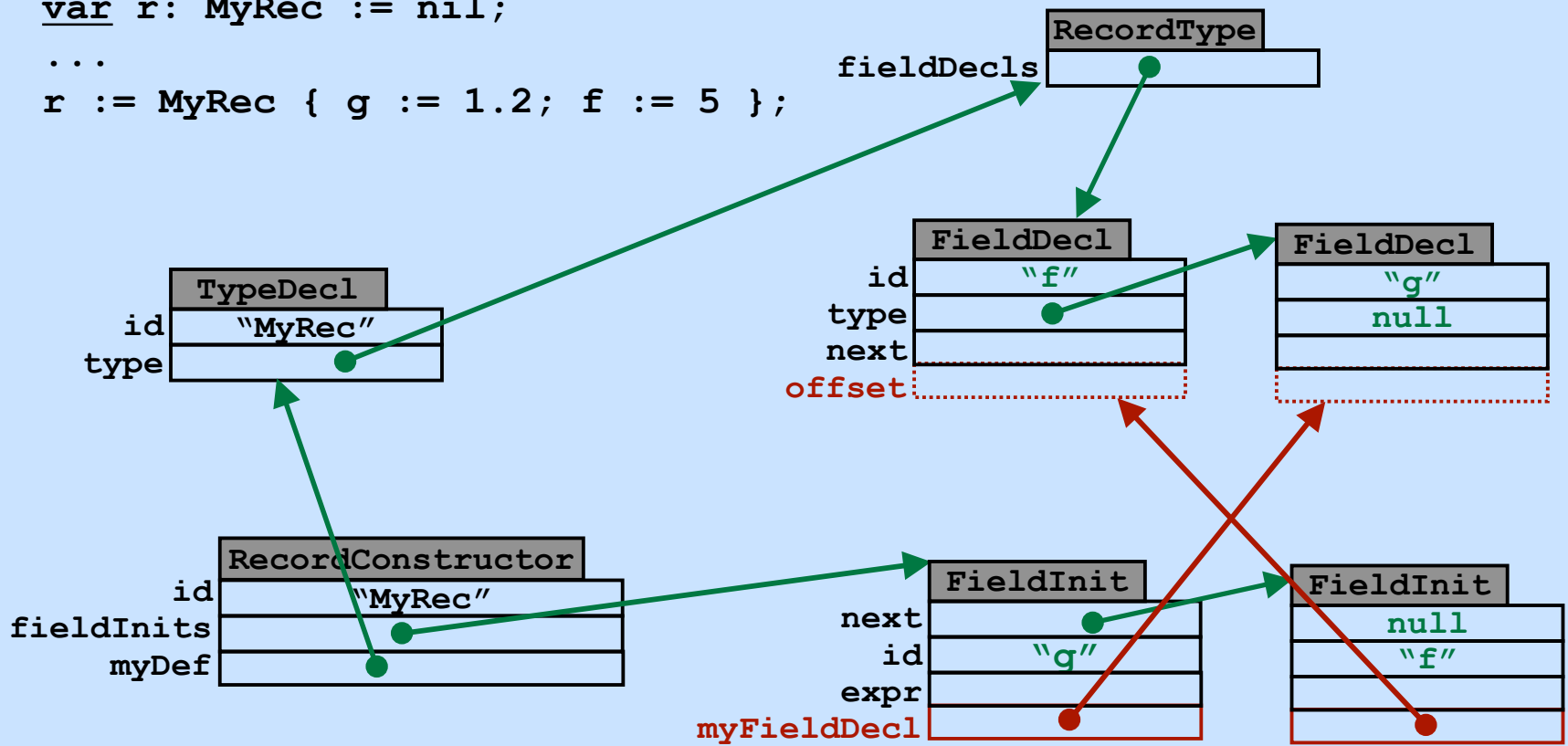
type MyRec is record
    f: integer;
    g: real;
end;
var r: MyRec := nil;
...
r := MyRec { g := 1.2; f := 5 };
    
```



Project 6 - Semantic Checking Part 2

New Field "myFieldDecl"

```
type MyRec is record  
    f: integer;  
    g: real;  
    end;  
var r: MyRec := nil;  
...  
r := MyRec { g := 1.2; f := 5 };
```



Project 6 - Semantic Checking Part 2

Testing

- OK to modify PrettyPrint.java

Add code to print out “mode” or “myProc” or “myLoop”

- OK to modify Main.java

Comment out the call to printAst

- OK to use your Lexer and Parser

- We’ll use the “standard” files in testing.

Make sure you test with standard files before submitting!

Project 6 - Semantic Checking Part 2

Testing

- PCAT Source programs without errors...

All output must agree exactly.

- PCAT Source program with errors...

`arrayOK.pcat`

`arrayErr.pcat`

Error messages (stderr) must agree exactly.

AST (stdout) will be ignored.

`run`

`runErr`

- Recommended Approach:

Get checker working with your own tests

THEN run my test suite.

If you pass them all... High confidence of program correctness!

Danger:

*Writing code just to handle
known example cases!*