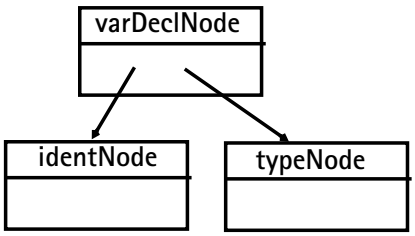
## Type Checking Simple Variable Declarations

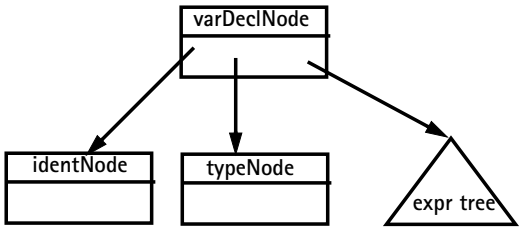



Type checking steps:

1. Check that **`identNode.idname`** is not already in the symbol table.
2. Enter **`identNode.idname`** into symbol table with **`type = typeNode.type`** and **`kind = Variable`**.



## Type Checking Initialized Variable Declarations




Type checking steps:

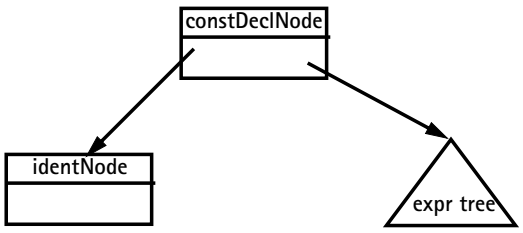
1. Check that **`identNode.idname`** is not already in the symbol table.
2. Type check initial value expression.
3. Check that the initial value's type is **`typeNode.type`**



4. Check that the initial value's kind is scalar (**`Variable`**, **`Value`** or **`ScalarParm`**).
5. Enter **`identNode.idname`** into symbol table with **`type = typeNode.type`** and **`kind = Variable`**.



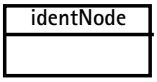
## Type Checking Const Decls




Type checking steps:

1. Check that **`identNode.idname`** is not already in the symbol table.
2. Type check the const value expr.
3. Check that the const value's kind is scalar (**`Variable`**, **`Value`** or **`ScalarParm`**).
4. Enter **`identNode.idname`** into symbol table with **`type`** = **`constValue.type`** and **`kind`** = **`Value`**.

## Type Checking IdentNodes

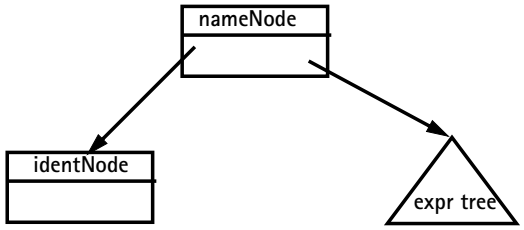



Type checking steps:

1. Lookup **`identNode.idname`** in the symbol table; error if absent.
2. Copy symbol table entry's **`type`** and **`kind`** information into the **`identNode`**.
3. Store a link to the symbol table entry in the **`identNode`** (in case we later need to access symbol table information).



## Type Checking NameNodes




Type checking steps:

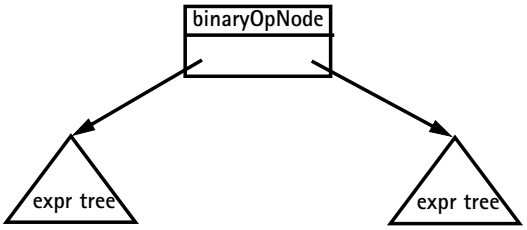
1. Type check the **`identNode`**.
2. If the **`subscriptVal`** is a null node, copy the **`identNode's type`** and **`kind`** values into the **`nameNode`** and return.
3. Type check the **`subscriptVal`**.
4. Check that **`identNode's kind`** is an array.



5. Check that **`subscriptVal's kind`** is scalar and **`type`** is integer or character.
6. Set the **`nameNode's type`** to the **`identNode's type`** and the **`nameNode's kind`** to **`Variable`**.



## Type Checking Binary Operators



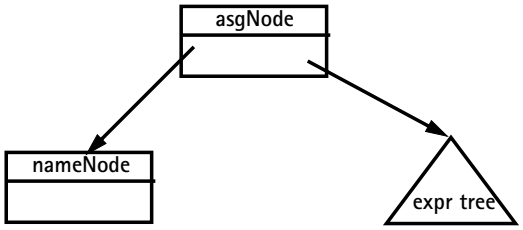

Type checking steps:

1. Type check left and right operands.
2. Check that left and right operands are both scalars.
3. **`binaryOpNode.kind`** = **`Value`**.

4. If **`binaryOpNode.operator`** is a plus, minus, star or slash:
   (a) Check that left and right operands have an arithmetic type (integer or character).
   (b) **`binaryOpNode.type`** = **`Integer`**
5. If **`binaryOpNode.operator`** is an and or is an or:
   (a) Check that left and right operands have a boolean type.
   (b) **`binaryOpNode.type`** = **`Boolean`**.
6. If **`binaryOpNode.operator`** is a relational operator:
   (a) Check that both left and right operands have an arithmetic type or both have a boolean type.
   (b) **`binaryOpNode.type`** = **`Boolean`**.



## Type Checking Assignments




Type checking steps:

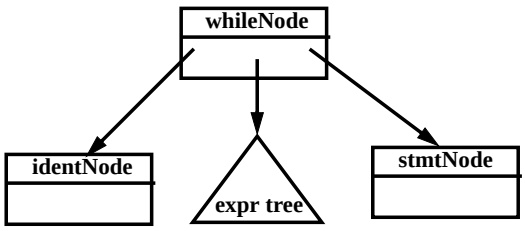
1. Type check the **`nameNode`**.
2. Type check the expression tree.
3. Check that the **`nameNode's kind`** is assignable (**`Variable`**, **`Array`**, **`ScalarParm`**, or **`ArrayParm`**).
4. If the **`nameNode's kind`** is scalar then check the expression tree's **`kind`** is also scalar and that both have the same type. Then return.



5. If the **`nameNode's`** and the expression tree's **`kinds`** are both arrays and both have the same **`type`**, check that the arrays have the same length. (Lengths of array parms are checked at run-time). Then return.
6. If the **`nameNode's kind`** is array and its **`type`** is character and the expression tree's kind is string, check that both have the same length. (Lengths of array parms are checked at run-time). Then return.
7. Otherwise, the expression may not be assigned to the **`nameNode`**.



## Type Checking While Loops



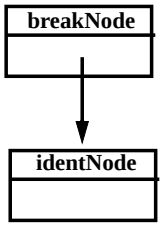

Type checking steps:

1. Type check the **`condition`** (an expr tree).
2. Check that the **`condition's type`** is **`Boolean`** and **`kind`** is scalar.
3. If the **`label`** is a null node then type check the **`stmtNode`** (the loop body) and return.

4.If there is a **`label`** (an **`identNode`**) then:
(a) Check that the **`label`** is not already present in the symbol table.
(b) If it isn't, enter **`label`** in the symbol table with **`kind=VisibleLabel`** and **`type= void`**.
(c) Type check the **`stmtNode`** (the loop body).
(d) Change the **`label's kind`** (in the symbol table) to **`HiddenLabel`**.



## Type Checking Breaks and Continues

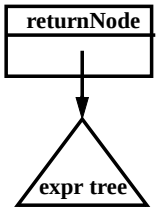



Type checking steps:

1. Check that the **`identNode`** is declared in the symbol table.
2. Check that **`identNode's kind`** is **`VisibleLabel`**. If **`identNode's kind`** is **`HiddenLabel`** issue a special error message.



## Type Checking Returns



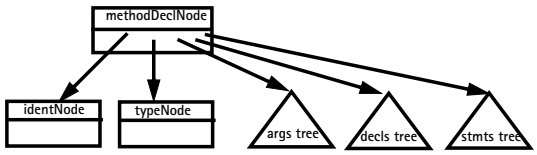

It is useful to arrange that a static field named ***`currentMethod`*** will always point to the **`methodDeclNode`** of the method we are currently checking.

Type checking steps:

1. If **`returnVal`** is a null node, check that **`currentMethod.returnType`** is **`Void`**.
2. If returnVal (an expr) is not null then check that returnVal's kind is scalar and returnVal's type is currentMethod.returnType.



## Type Checking Method Declarations



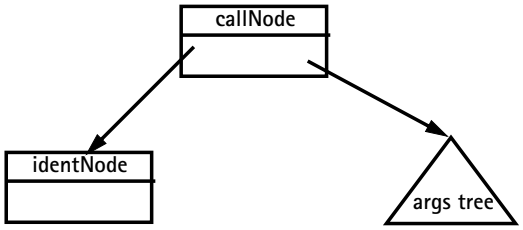

Type checking steps:

1. Create a new symbol table entry **`m`**, with **`type`** = **`typeNode.type`** and **`kind`** = **`Method`**.
2. Check that **`identNode.idname`** is not already in the symbol table; if it isn't, enter **`m`** using **`identNode.idname`**.
3. Create a new scope in the symbol table.
4. Set **`currentMethod`** = this **`methodDeclNode`**.

5. Type check the **`args`** subtree.
6. Build a list of the symbol table nodes corresponding to the **`args`** subtree; store it in **`m`**.
7. Type check the **`decls`** subtree.
8. Type check the **`stmts`** subtree.
9. Close the current scope at the top of the symbol table.



## Type Checking Method Calls




We consider calls of procedures in a statement. Calls of functions in an expression are very similar.

Type checking steps:

1. Check that **`identNode.idname`** is declared in the symbol table. Its type should be **`Void`** and kind should be **`Method`**.
2. Type check the **`args`** subtree.
3. Build a list of the expression nodes found in the args subtree.



4. Get the list of parameter symbols declared for the method (stored in the method's symbol table entry).
5. Check that the arguments list and the parameter symbols list both have the same length.
6. Compare each argument node with its corresponding parameter symbol:
   (a) Both should have the same type.
   (b) A **`Variable`**, **`Value`**, or **`ScalarParm`** kind in an argument node matches a **`ScalarParm`** parameter. An **`Array`** or **`ArrayParm`** kind in an argument node matches an **`ArrayParm`** parameter.