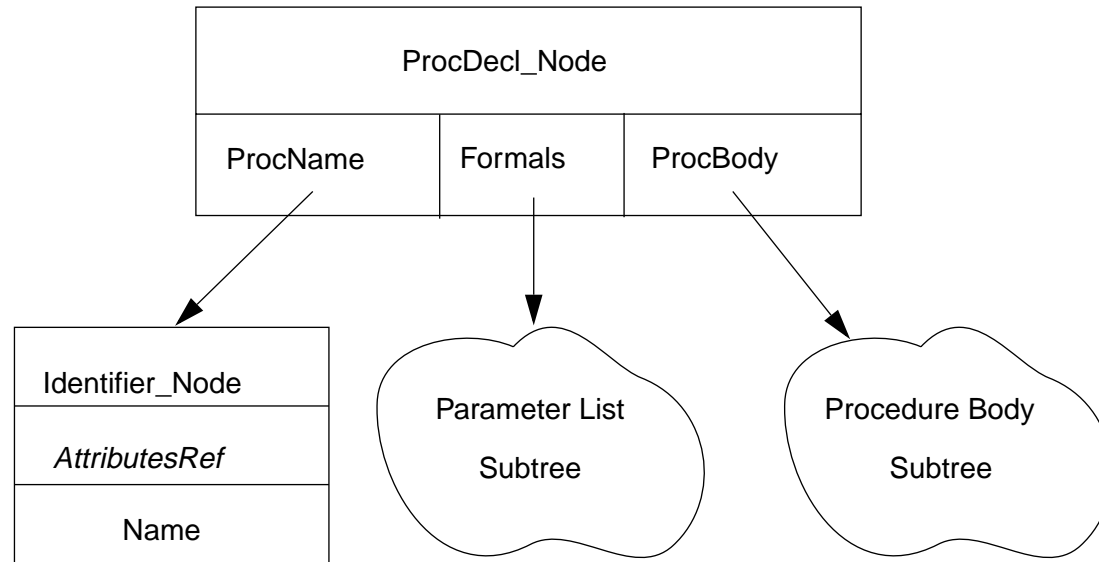


Procedure Declarations



`ProcDecl_Node.Semantics()`

Enter `ProcName.Name` into the symbol table

If it is already there, check that it is a procedure id and that its `BodyDeclared` attribute is `False`

else Create an `Attributes` record for this name indicating

- it is a procedure id (a new id type)
- `NestingLevel = CurrentProc.NestingLevel + 1`
- `LocalDecls = a new symbol table`
- `Parameters = null`

Save a reference to the `Attributes` (as usual) record in the `Identifier_Node`

```
ProcDecl_Node.Semantics ( ) -- contin
```

```
[Local variable: ContainingProc]
```

```
...
```

```
Set ContainingProc to the value of CurrentProc
```

```
Set CurrentProc to reference the Attributes record of this procedure
```

```
Make CurrentProc.LocalDecls the current symbol table scope
```

```
-- If parameter declarations a present, call Formals.Semantics() here
```

```
If ProcBody = null then
```

```
    Set BodyDeclared in the Attributes record to False
```

```
else
```

```
    Set BodyDeclared in the Attributes record to True
```

```
    ProcBody.Semantics()
```

```
Restore the previous symbol table scope
```

```
Restore the value of CurrentProc from ContainingProc
```

Formals_Node.Semantics ()

The parameters declarations are in the AST as a list of Formals_Nodes. The following routine is used to process each.

Formals_Node.Semantics():

- Call ParamType.TypeSemantics()

- For each Id on ParamIdList

 - Enter it in the current symbol table

 - Create an ParameterAttributes record for it with

 - Mode derived from ParamMode

 - Level = CurrentProc.NestingLevel

 - Offset = CurrentProc.FrameSize

 - Allocate space by incrementing CurrentProc.FrameSize as appropriately for the mode and type of the parameter

 - Link the parameter onto the CurrentProc.Parameters list

Code Generation for Procedure Declarat

ProcDecl_Node.Code_Gen():

 If the StartLabel field of the T's Attributes record does not have a value

 Set it to a new label value

 If T.ProcBody is not null then

 Set SaveCurrentProc to the value of CurrentProc

 Set CurrentProc to reference the Attributes record of this procedure

 EndLabel <- a new label

 Generate a branch to EndLabel

 Call Code_Gen(T.ProcBody)

 Generate the label EndLabel

 Restore the value of CurrentProc from SaveCurrentProc

ProcBody_Node.Code_Gen():

 Generate the label CurrentProc.StartLabel

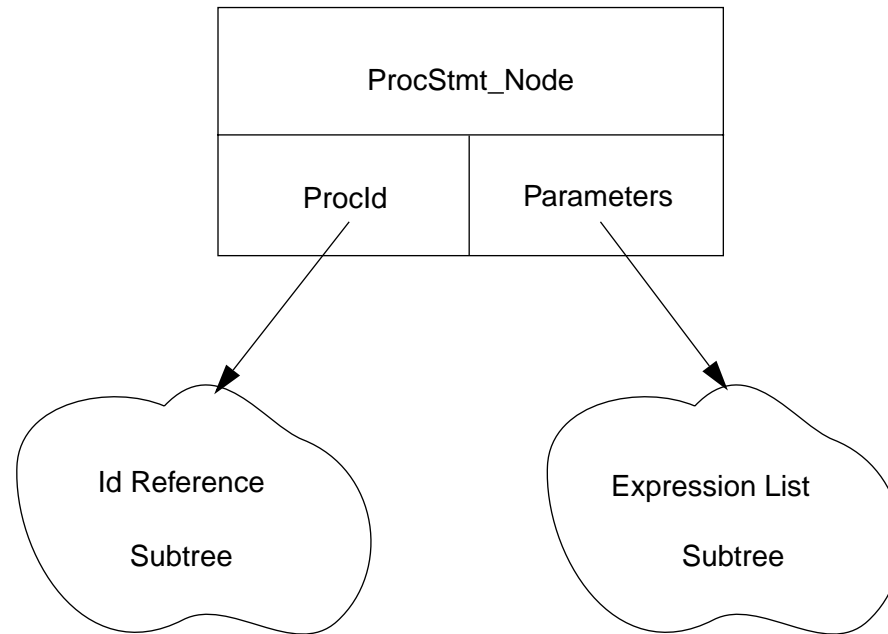
 Generate procedure entry code

 Call Code_Gen(T.ProcBodyDecl_List)

 Call Code_Gen(T.ProcBodyStmt_List)

 Generate procedure exit cod

Procedure Calls without Parameters



`ProcStmt_Node.Semantics()`:

`Procid.Semantics()`

Check that `Procid.AttributesRef` designates a procedure

Procedure Calls with Parameters

ProcStmt_Node.Semantics():

Semantics(ProclD)

Check that ProclD designates a procedure

If this procedure has parameters, walk down the tree referenced by ProclD.Parameters, checking parameter type compatibility and number of parameters

ProcStmt_Node.GenCode():

If this procedure has parameters, walk down the tree referenced by ProcStmtId.Parameters. For each parameter, generate code to load its value or address into the appropriate location (register or storage)

Generate a jump and link instruction to the StartLabel of the procedure designated by ProcStmtId

Variables and Recursive Procedures

in ProcDecl_Node.Semantics():

When creating the Attributes record for the procedure, include field
FrameSize initialized to a value that allows space for saved registers
(8 will be sufficient in your compiler, allowing space to save
\$ra and a display register)

in VarDecl_Node.Semantics():

When allocating space for each variable, its Offset will be taken from
CurrentProc.FrameSize and that value will be incremented by the
size of the type of the variable

A new attribute, Level, must be saved for each variable. The value of
Level is set to CurrentProc.NestingLevel

in the code generation routine Get_Variable_Address

When addressing a variable, its Level attribute designates a display
register whose value is added to Offset in order to address the variable

Implementing Parameter Modes

In paramaters:

- not assignable within a procedure body
- can be implemented by copying actual parameter or passing an
- use copying for your implementation

In out paramaters:

- carry values into and out of procedures
- the actual paramater must be assignable
- can be implemented by copy in-copy out or passing an address
- use address passing for your implementation

Out paramaters:

- carry value out of procedures
- the actual paramater must be assignable
- can be implemented by copy out or passing an address
- use address passing for your implementation