PCAT Language Differences

Keywords are now lowercase

ELSIF --> elseif

Punctuation for array initialization

Tolmach:
VAR A: MyArray := MyArray { 7, 9, 3 of 11, 13, 15 };

Porter:
var a: MyArray := MyArray {{ 7, 9, 3 of 11, 13, 15 }};

Relational Operators

Tolmach:
a = b = c  syntax error

Porter:
a = b = c  okay
(a = b) = c  (equivalent)

Recursive Types

Tolmach:
TYPE T1 IS RECORD
  val: INTEGER;
  next: T2;
END;

AND
  T2 IS RECORD
    val: INTEGER;
    next: T1;
  END;

Porter:
type T1 is record
  var: integer;
  next: T2;
end;
T2 is record
  var: integer;
  next: T1;
end;
Recursive Types

Porter:  
```plaintext
type T1 is record  
  var: integer;  
  next: T2;  
end;  
T2 is record  
  var: integer;  
  next: T1;  
end;
```

Equivalent to:  
```plaintext
type T1 is record  
  var: integer;  
  next: T2;  
end;  
type T2 is record  
  var: integer;  
  next: T1;  
end;
```

Recursive Procedures

Tolmach:  
```plaintext
PROCEDURE foo (...) IS BEGIN ... bar(); ... END;  
AND  
  bar (...) IS BEGIN ... foo(); ... END;
```

Porter:  
```plaintext
procedure  
  foo (...) is begin ... bar(); ... end;  
  bar (...) is begin ... foo(); ... end;
```

Equivalent to:  
```plaintext
procedure foo (...) is begin ... bar(); ... end;  
procedure bar (...) is begin ... foo(); ... end;
```
Differences in AST

Class names, field names

Additional (non-syntactic fields)
  Added / filled in during type-checking

Lists vs. Arrays
  Example: formal parameters

List vs. Arrays

Example: formal parameters

Tolmach:
  public static class ProcDec extends Dec {
  ...
    FormalParam[] formals;
  ...
  }

Porter:
  static class ProcDecl extends Node {
  ...
    Formal       formals;
  ...
  }

  static class Formal extends Node {
  ...
    Formal       next;
  }
**List vs. Arrays**

Going through the list...

*Tolmach:*

```java
for (int i = 0; i < formals.length; i++) {
    ... formals[i] ...
}
```

*Porter:*

```java
for (Formal f = formals; f = f.next; f) {
    ... f ...
}
```

---

**Statement Sequences**

*Tolmach:*

```java
public abstract static class St extends Node {
    ... (no fields)...
}
public static class WhileSt extends St {
    ...
    St body;
    ...
}
public static class SequenceSt extends St {
    ...
    St[] statements;
    ...
}
```

© Harry H. Porter, 2004
Statement Sequences

Porter:
Each statement contains a “next” pointer
Linked lists of statements
Anywhere a single statement can appear

abstract static class Stmt extends Node {
    Stmt next;
}
static class WhileStmt extends Stmt {
    ...
    Stmt stmts;
    ...
}

Checker Class

Class name: “Checker”
Single instance
Routines:
    checkIfStmt
    checkBinaryOp
    ...

Class name: “Generator”
The “main” Method

Ast.Body ast;
Parser parser;
Checker checker;
...
// Parse the source and return the AST.
parser = new Parser (args);
ast = parser.parseProgram ();

// Check the AST.
checker = new Checker ();
checker.checkAst (ast);

The “IntToReal” Class

var i: integer;
    x: real;
...
r := i;

Inserted into AST
During type checking
Indicates where a data conversion is required

static class IntToReal extends Expr {
    Expr expr;
}

© Harry H. Porter, 2004