The Visitor Pattern and Compiler Phases

CS 4447 / CS 9545 – Stephen M. Watt

***The* University *of* Western Ontario**

Non-OO Compiler Implementations

Each phase is structured,e.g., as:

```
void
tiBottomUp(Stab stab, AbSyn absyn, TForm type)
{
        Scope("tiBottomUp");

        TPoss      fluid(tuniReturnTPoss);     TPoss      fluid(tuniYieldTPoss);
        TForm      fluid(tuniYieldType);       TPoss      fluid(tuniExitTPoss);
        int        fluid(tloopBreakCount);
        SymbolList fluid(terrorIdComplaints);
        AbLogic    fluid(abCondKnown);
        tuniYieldTPoss     = tuniInappropriateTPoss; tuniYieldType      = tfUnknown;
        tuniReturnTPoss    = tuniInappropriateTPoss;
        tuniExitTPoss      = tuniInappropriateTPoss; tuniExitType       = tfUnknown;
        tloopBreakCount    = -1;
        terrorIdComplaints = 0;
        abCondKnown        = abCondKnown ? ablogCopy(abCondKnown) : ablogTrue();

        tibup(stab, absyn, type);

        listFree(Symbol)(terrorIdComplaints);
        ablogFree(abCondKnown);

        ReturnNothing;
}
```

Non-OO Compiler Implementations

```
void
tibup(Stab stab, AbSyn absyn, TForm type)
{
        assert(absyn);

        /* Check before processing the stab, if present. */
        if (abState(absyn) >= AB_State_HasPoss)
                return;
        . . .

        switch (abTag(absyn)) {
        case AB_Id:           tibupId         (stab, absyn, type); break;
        case AB_LitInteger:   tibupLitInteger (stab, absyn, type); break;
        case AB_Apply:        tibupApply      (stab, absyn, type); break;
        . . .
        case AB_While:        tibupWhile      (stab, absyn, type); break;
        case AB_With:         tibupWith       (stab, absyn, type); break;
        case AB_Yield:        tibupYield      (stab, absyn, type); break;
        default:              bugBadCase      (abTag(absyn));
        }

        . . .
}
```

Non-OO Compiler Implementations

```
local void
tibupApply(Stab stab, AbSyn absyn, TForm type)
{
        AbSyn           op = abApplyOp(absyn);
        TPoss           tp;

        tibup(stab, op, tfUnknown);

        tp = abReferTPoss(op);

        if (tpossHasMapType(tp) || tpossCount(tp) == 0)
                tibup0ApplyFType(stab, absyn, type,
                                 op, abApplyArgc(absyn), abApplyArgf);
        else
                tibup0ApplySym(stab, absyn, type,
                               ssymApply, abArgc(absyn), abArgf, NULL);

        tpossFree(tp);
}
```

OO Compiler Phases as Visitors

```
public abstract class CCode {
  public void acceptVisitor(CCodeVisitor v) { v.visit(this); }
  ...
}
...
abstract class CCodeStat extends CCode { }

abstract class CCodeExpr extends CCode {
  CType       _optTypeAnnotation = null;
}
...
class CCodeStatCompound extends CCodeStat {
  public void acceptVisitor(CCodeVisitor v) { v.visit(this); }
}
...
class CCodeExprInfix extends CCodeExpr {
  public void acceptVisitor(CCodeVisitor v) { v.visit(this); }
}
```

OO Compiler Phases as Visitors

```
public interface CCodeVisitor {
  // Top Level
  public void visit(CCode cc);
  public void visit(CCodeUnit cc);
  public void visit(CCodeFunction cc);
  public void visit(CCodeDeclaration cc);

  // Statements
  public void visit(CCodeStatCompound cc);
  …
  // Expressions
  public void visit(CCodeExprInfix cc);
  …
}
```

OO Compiler Phases as Visitors

- Symbol table construction
- Type inference
- Code generation
- Structure saving/printing…

CCodePrinter as a Visitor

```
package com.smwatt.comp;

import java.io.PrintStream;
import java.util.List;

public class CCodePrinter implements CCodeVisitor {
    TabbingStream    _tout;
    boolean          _isDebugMode = false;


    CCodePrinter(TabbingStream out) {
        _tout = out;
    }
    CCodePrinter(PrintStream out) {
        _tout = new TabbingStream(out);
    }
    public void print(CCode cc) { visit(cc); }

    ...
}
```

CCodePrinter as a Visitor II

```
public class CCodePrinter implements CCodeVisitor {
    ...
    //-- Helper methods ------------------------------------------------
    private void printSep(List<? extends CCode> lcc, String sep) {
        int i = 0;
        for (CCode cc: lcc) {
            if (i++ > 0) _tout.print(sep);
            visit(cc);
        }
    }
    private void printIndented(List<? extends CCode> lcc, String sep, String fin) {
        _tout.incIndent();
        int i = 0, sz = lcc.size();
        for (CCode cc: lcc) {
            _tout.printlnIndent();
            visit(cc);
            _tout.print(++i < sz ? sep : fin);
        }
        _tout.decIndent();
        _tout.printlnIndent();
    }
    private void printStatIndented(CCodeStat cc) { … }
    ...
}
```

CCodePrinter as a Visitor III

```
public class CCodePrinter implements CCodeVisitor {
    ...
    ////////////////////////////////////////////////////////////////////
    //
    // Declarators
    //
    ////////////////////////////////////////////////////////////////////
    public void visit(CCodeDeclaratorArray cc) {
        if (cc._optAr != null) visit(cc._optAr);
        _tout.print("[");
        if (cc._optIndex != null) visit(cc._optIndex);
        _tout.print("]");
    }
    public void visit(CCodeDeclaratorFunction cc) {
        if (cc._optFn != null) visit(cc._optFn);
        _tout.print("(");
        printSep(cc._argl, ", ");
        _tout.print(")");
    }
    public void visit(CCodeDeclaratorInit cc) {
        visit(cc._dtor);
        _tout.print(" = ");
        visit(cc._initializer);
    }
    ...
```

CCodePrinter as a Visitor IV

```
public class CCodePrinter implements CCodeVisitor {
    ...
    ////////////////////////////////////////////////////////////////////
    //
    // Statements
    //
    ////////////////////////////////////////////////////////////////////

    public void visit(CCodeStatBreak cc) {
        _tout.print("break;");
    }
    public void visit(CCodeStatCase cc) {
        _tout.print("case ");
        visit(cc._value);
        _tout.print(":");
        if (cc._stat instanceof CCodeStatCase    ||
            cc._stat instanceof CCodeStatDefault
        )
            _tout.printlnIndentRelative(-1);
        else
            _tout.tabTo(_tout.getCurrentIndent());
        visit(cc._stat);
    }
    public void visit(CCodeStatCompound cc) { ... }
   . . .
```

CCodePrinter as a Visitor V

```
public class CCodePrinter implements CCodeVisitor {
    ...
    ////////////////////////////////////////////////////////////////////
    //
    // Expressions
    //
    ////////////////////////////////////////////////////////////////////

    public void visit(CCodeExprAssignment cc) {
        visit(cc._a);
        _tout.print(" ");
        _tout.print(cc._op.toString());
        _tout.print(" ");
        visit(cc._b);
    }
    public void visit(CCodeExprCast cc) {
        _tout.print("(");
        visit(cc._typename);
        _tout.print(") ");
        visit(cc._expr);
    }
    ...
```

CCodePrinter as a Visitor VI

```
public class CCodePrinter implements CCodeVisitor {
    ...
    public void visit(CCodeExprConditional cc) {
        visit(cc._test);
        _tout.print(" ? ");
        visit(cc._thexpr);
        _tout.print(" : ");
        visit(cc._elexpr);
    }
    public void visit(CCodeExprInfix cc) {
        visit(cc._a);
        _tout.print(" ");
        _tout.print(cc._op.toString());
        _tout.print(" ");
        visit(cc._b);
    }
    public void visit(CCodeExprParen cc) {
        _tout.print("(");
        visit(cc._expr);
        _tout.print(")");
    }
    ...
}
```

Keeping Track of Where You Are

- Some traversals need to keep track of what kind of branch they are in.
  - E.g. L-value context vs R-value context (for code generation).

- Can keep track of this with a state field in the visitor.
  (Like the tabbing stream was a state variable in the CCodePrinter visitor.)