CMPE 152: Compiler Design

October 26 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

SILICON VALLEY'S FIRST CHOICE FOR NEW ENGINEERING HIRES
— Silicon Valley Business Journal, 2013

San José State UNIVERSITY

Parse Tree Processing

- Recall that after the frontend parser builds the parse tree, it's the backend that processes it.
- ANTLR provides utilities to process the parse tree.
- ANTLR can generate code to process a parse tree with two types of tree walkers:
  - listener interface (default)
  - visitor interface

Parse Tree Listener Interface

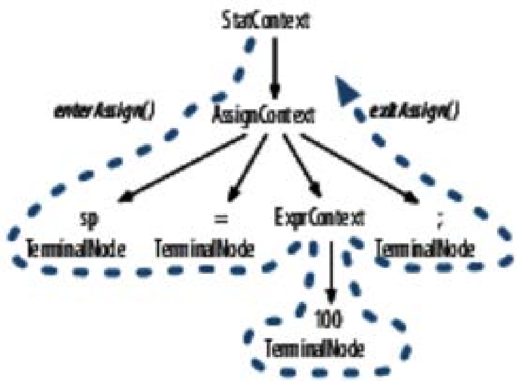
- ANTLR generates code that automatically performs a <u>depth-first walk</u> of the parse tree.
  - You do not have to write a tree walker.

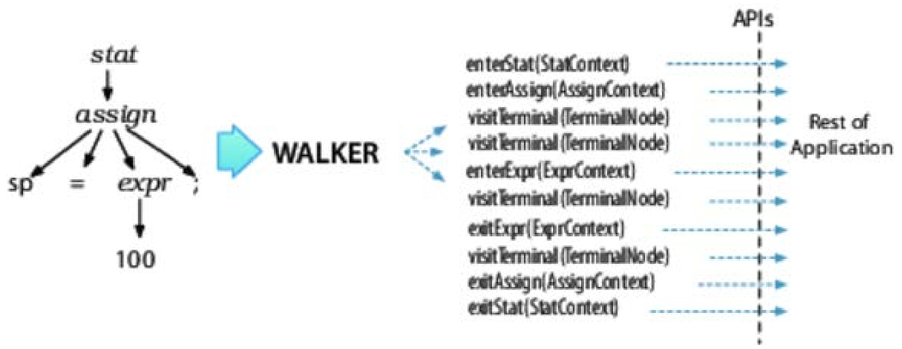
- It generates a `ParseTreeListener` subclass that is specific to each grammar.
  - The listener class has default <u>enter and exit methods</u> for each rule.

- You write a subclass that overrides the default <u>enter and exit methods</u> to do what you want.
  - You do not have to explicitly visit child nodes.

Parse Tree Listener Interface, *cont'd*

- Tree walk and call sequence:





```
enterStat(StatContext)
enterAssign(AssignContext)
visitTerminal(TerminalNode)
visitTerminal(TerminalNode)
enterExpr(ExprContext)
visitTerminal(TerminalNode)
exitExpr(ExprContext)
visitTerminal(TerminalNode)
exitAssign(AssignContext)
exitStat(StatContext)
```

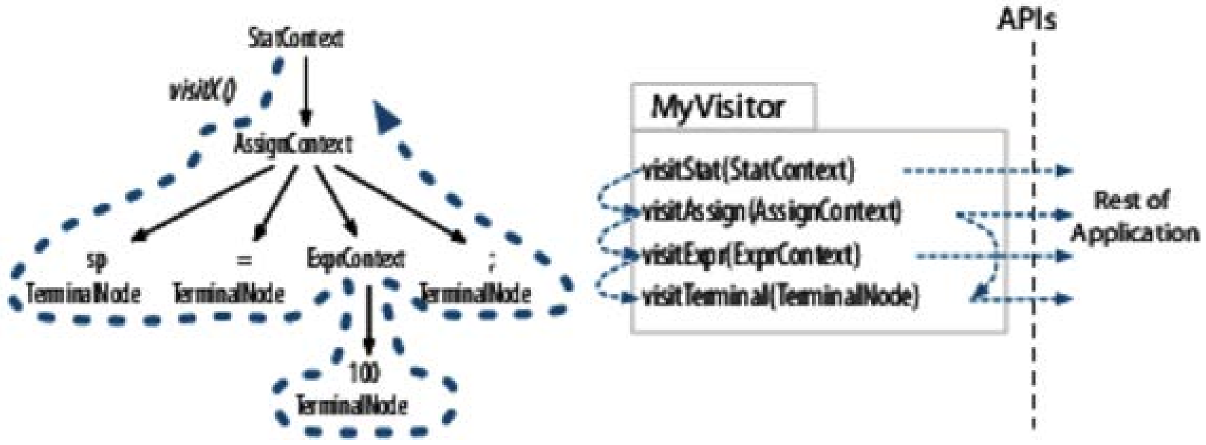
Parse Tree Visitor Interface

- The visitor interface give you more control over the tree walk.
  - Specify **`-visitor`** on the **`antlr4`** command.
  - Also **`-no-listener`**

```
antlrr4 -no-listener -visitor LabeledExpr.g4
```

- ANTLR generates a grammar-specific <u>visitor interface</u> and a <u>visitor method</u> per rule.
- You must <u>initiate the visit</u> by calling **`visit()`** on the parse tree root.

Parse Tree Visitor Interface, *cont'd*



Visitor Interface Example

- Interpret arithmetic expressions by computing values in the back end.
- Label each rule alternative.
- ANTLR generates a different visitor per labeled alternative.
  - Default: ANTLR generates only one visitor per rule.

Visitor Interface Example, *cont'd*

LabeledExpr.g4

```
grammar LabeledExpr;

prog:   stat+ ;

stat:   expr NEWLINE                # printExpr
    |   ID '=' expr NEWLINE         # assign
    |   NEWLINE                     # blank
    ;

expr:   expr op=('*'|'/') expr      # MulDiv
    |   expr op=('+'|'-') expr      # AddSub
    |   INT                         # int
    |   ID                          # id
    |   '(' expr ')'                # parens
    ;

MUL :   '*' ; // assigns token name to '*' used above in grammar
DIV :   '/' ;
ADD :   '+' ;
SUB :   '-' ;
ID  :   [a-zA-Z]+ ;      // match identifiers
INT :   [0-9]+ ;         // match integers
NEWLINE:'\r'? '\n' ;     // return newlines to parser (is end-statement signal)
WS  :   [ \t]+ -> skip ; // toss out whitespace
```

Labeled rules

Named tokens

Visitor Interface Example, *cont'd*

- ANTLR-generated visitor interface
  - Uses Java generics to handle different result types.

LabeledExprVisitor.java

```
public interface LabeledExprVisitor<T> extends ParseTreeVisitor<T>
{
    T visitProg(LabeledExprParser.ProgContext ctx);
    T visitPrintExpr(LabeledExprParser.PrintExprContext ctx);
    T visitAssign(LabeledExprParser.AssignContext ctx);
    T visitBlank(LabeledExprParser.BlankContext ctx);
    T visitParens(LabeledExprParser.ParensContext ctx);
    T visitMulDiv(LabeledExprParser.MulDivContext ctx);
    T visitAddSub(LabeledExprParser.AddSubContext ctx);
    T visitId(LabeledExprParser.IdContext ctx);
    T visitInt(LabeledExprParser.IntContext ctx);
}
```

Visitor Interface Example, *cont'd*

- Default implementation: During each visit, just visit the children.

```
public class LabeledExprBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements LabeledExprVisitor<T>
{
    @Override public T visitProg(LabeledExprParser.ProgContext ctx) { return visitChildren(ctx); }
    @Override public T visitPrintExpr(LabeledExprParser.PrintExprContext ctx) { return visitChildren(ctx); }
    @Override public T visitAssign(LabeledExprParser.AssignContext ctx) { return visitChildren(ctx); }
    @Override public T visitBlank(LabeledExprParser.BlankContext ctx) { return visitChildren(ctx); }
    @Override public T visitParens(LabeledExprParser.ParensContext ctx) { return visitChildren(ctx); }
    @Override public T visitMulDiv(LabeledExprParser.MulDivContext ctx) { return visitChildren(ctx); }
    @Override public T visitAddSub(LabeledExprParser.AddSubContext ctx) { return visitChildren(ctx); }
    @Override public T visitId(LabeledExprParser.IdContext ctx) { return visitChildren(ctx); }
    @Override public T visitInt(LabeledExprParser.IntContext ctx) { return visitChildren(ctx); }
}
```

LabeledExprBaseVisitor.java

Visitor Interface Example, *cont'd*

- Override the pertinent visitor methods.

EvalVisitor.java

```
public class EvalVisitor extends LabeledExprBaseVisitor<Integer>
{
    /** "memory" for our calculator; variable/value pairs go here */
    Map<String, Integer> memory = new HashMap<String, Integer>();

    /** ID '=' expr NEWLINE */
    @Override
    public Integer visitAssign(LabeledExprParser.AssignContext ctx)
    {
        String id = ctx.ID().getText();  // id is left-hand side of '='
        int value = visit(ctx.expr());   // compute value of expression on right
        memory.put(id, value);           // store it in our memory
        return value;
    }

    ...
```

Visitor Interface Example, *cont'd*

EvalVisitor.java

```
/** expr NEWLINE */
@Override
public Integer visitPrintExpr(LabeledExprParser.PrintExprContext ctx)
{
    Integer value = visit(ctx.expr()); // evaluate the expr child
    System.out.println(value);         // print the result
    return 0;                          // return dummy value
}

/** INT */
@Override
public Integer visitInt(LabeledExprParser.IntContext ctx)
{
    return Integer.valueOf(ctx.INT().getText());
}

/** ID */
@Override
public Integer visitId(LabeledExprParser.IdContext ctx)
{
    String id = ctx.ID().getText();
    if ( memory.containsKey(id) ) return memory.get(id);
    return 0;
}
```

Visitor Interface Example, *cont'd*

EvalVisitor.java

```
    /** expr op=('*'|'/') expr */
    @Override
    public Integer visitMulDiv(LabeledExprParser.MulDivContext ctx)
    {
        int left = visit(ctx.expr(0));  // get value of left subexpression
        int right = visit(ctx.expr(1)); // get value of right subexpression
        if ( ctx.op.getType() == LabeledExprParser.MUL ) return left * right;
        return left / right; // must be DIV
    }

    /** expr op=('+'|'-') expr */
    @Override
    public Integer visitAddSub(LabeledExprParser.AddSubContext ctx)
    {
        int left = visit(ctx.expr(0));  // get value of left subexpression
        int right = visit(ctx.expr(1)); // get value of right subexpression
        if ( ctx.op.getType() == LabeledExprParser.ADD ) return left + right;
        return left - right; // must be SUB
    }
```

Visitor Interface Example, *cont'd*

EvalVisitor.java

```
    /** '(' expr ')' */
    @Override
    public Integer visitParens(LabeledExprParser.ParensContext ctx)
    {
        return visit(ctx.expr()); // return child expr's value
    }
}
```

Visitor Interface Example, *cont'd*

- The main program:

Calc.java

```
public class Calc
{
    public static void main(String[] args) throws Exception
    {
        String inputFile = null;

        if (args.length > 0) inputFile = args[0];
        InputStream is = (inputFile != null)
                                ? new FileInputStream(inputFile)
                                : System.in;

        ANTLRInputStream input = new ANTLRInputStream(is);
        LabeledExprLexer lexer = new LabeledExprLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        LabeledExprParser parser = new LabeledExprParser(tokens);
        ParseTree tree = parser.prog(); // parse

        EvalVisitor eval = new EvalVisitor();
        eval.visit(tree);
    }
}
```

Demo

Visitor Interface Example, *cont'd*

- Corresponding C++ code:

```
class LabeledExprVisitor : public antlr4::tree::AbstractParseTreeVisitor
{
public:
    virtual antlrcpp::Any visitProg(LabeledExprParser::ProgContext *context) = 0;
    virtual antlrcpp::Any visitPrintExpr(LabeledExprParser::PrintExprContext *context) = 0;
    virtual antlrcpp::Any visitAssign(LabeledExprParser::AssignContext *context) = 0;
    virtual antlrcpp::Any visitBlank(LabeledExprParser::BlankContext *context) = 0;
    virtual antlrcpp::Any visitParens(LabeledExprParser::ParensContext *context) = 0;
    virtual antlrcpp::Any visitMulDiv(LabeledExprParser::MulDivContext *context) = 0;
    virtual antlrcpp::Any visitAddSub(LabeledExprParser::AddSubContext *context) = 0;
    virtual antlrcpp::Any visitId(LabeledExprParser::IdContext *context) = 0;
    virtual antlrcpp::Any visitInt(LabeledExprParser::IntContext *context) = 0;
};
```

LabeledExprVisitor.h

Visitor Interface Example, *cont'd*

- Default implementation: During each visit, just visit the children.

LabeledExprBaseVisitor.h

```
class  LabeledExprBaseVisitor : public LabeledExprVisitor
{
public:
  virtual antlrcpp::Any visitProg(LabeledExprParser::ProgContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitPrintExpr(LabeledExprParser::PrintExprContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitAssign(LabeledExprParser::AssignContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitBlank(LabeledExprParser::BlankContext *ctx) override
  {
    return visitChildren(ctx);
  }
```

Visitor Interface Example, *cont'd*

```
  virtual antlrcpp::Any visitParens(LabeledExprParser::ParensContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitMulDiv(LabeledExprParser::MulDivContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitAddSub(LabeledExprParser::AddSubContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitId(LabeledExprParser::IdContext *ctx) override
  {
    return visitChildren(ctx);
  }

  virtual antlrcpp::Any visitInt(LabeledExprParser::IntContext *ctx) override
  {
    return visitChildren(ctx);
  }
};
```

LabeledExprBaseVisitor.h

Visitor Interface Example, *cont'd*

- Override the pertinent visitor methods.

EvalVisitor.h

```
class EvalVisitor : public LabeledExprBaseVisitor
{
public:
    antlrcpp::Any visitAssign(LabeledExprParser::AssignContext *ctx) override;
    antlrcpp::Any visitPrintExpr(LabeledExprParser::PrintExprContext *ctx) override;
    antlrcpp::Any visitInt(LabeledExprParser::IntContext *ctx) override;
    antlrcpp::Any visitId(LabeledExprParser::IdContext *ctx) override;
    antlrcpp::Any visitMulDiv(LabeledExprParser::MulDivContext *ctx) override;
    antlrcpp::Any visitAddSub(LabeledExprParser::AddSubContext *ctx) override;
    antlrcpp::Any visitParens(LabeledExprParser::ParensContext *ctx) override;

private:
    map<string, int> memory;
};
```

Visitor Interface Example, *cont'd*

```
antlrcpp::Any EvalVisitor::visitAssign(LabeledExprParser::AssignContext *ctx)
{
    string id = ctx->ID()->getText();
    int value = visit(ctx->expr());
    memory[id] = value;
    return value;
}

antlrcpp::Any EvalVisitor::visitPrintExpr(LabeledExprParser::PrintExprContext *ctx)
{
    int value = visit(ctx->expr());
    cout << value << endl;
    return 0;
}

antlrcpp::Any EvalVisitor::visitInt(LabeledExprParser::IntContext *ctx)
{
    return stoi(ctx->INT()->getText());
}
```

EvalVisitor.cpp

Visitor Interface Example, *cont'd*

EvalVisitor.cpp

```
antlrcpp::Any EvalVisitor::visitId(LabeledExprParser::IdContext *ctx)
{
    string id = ctx->ID()->getText();
    return (memory.find(id) != memory.end())
    ? memory[id] : 0;
}

antlrcpp::Any EvalVisitor::visitMulDiv(LabeledExprParser::MulDivContext *ctx)
{
    int left  = visit(ctx->expr(0));
    int right = visit(ctx->expr(1));
    return (ctx->op->getType() == LabeledExprParser::MUL)
            ? left*right : left/right;
}
```

Visitor Interface Example, *cont'd*

EvalVisitor.cpp

```
antlrcpp::Any EvalVisitor::visitAddSub(LabeledExprParser::AddSubContext *ctx)
{
    int left  = visit(ctx->expr(0));
    int right = visit(ctx->expr(1));
    return (ctx->op->getType() == LabeledExprParser::ADD)
            ? left + right : left - right;
}

antlrcpp::Any EvalVisitor::visitParens(LabeledExprParser::ParensContext *ctx)
{
    return visit(ctx->expr());
}
```

Visitor Interface Example, *cont'd*

- The main program:

Calc.cpp

```
int main(int argc, const char *args[])
{
    ifstream ins;
    ins.open(args[1]);

    ANTLRInputStream input(ins);
    LabeledExprLexer lexer(&input);
    CommonTokenStream tokens(&lexer);

    LabeledExprParser parser(&tokens);
    tree::ParseTree *tree = parser.prog();

    cout << "Parse tree:" << endl;
    cout << tree->toStringTree(&parser) << endl;

    cout << endl << "Evaluation:" << endl;
    EvalVisitor eval;
    eval.visit(tree);

    delete tree;
    return 0;
}
```

Demo