Plan for Today

Abstract Syntax Tree

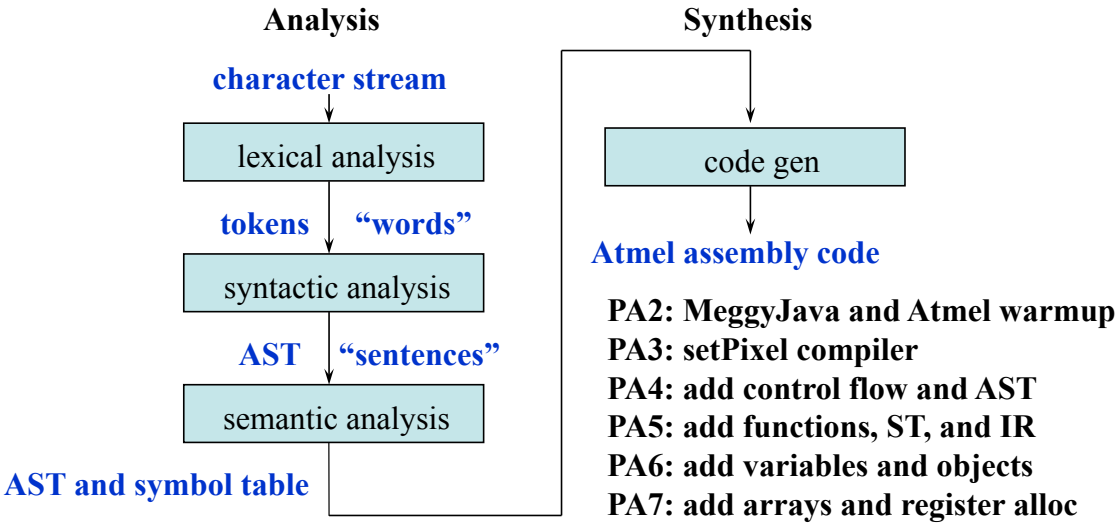
- Example and main idea
- construction with a bottom up parser
- AST provided for PA4 through PA7

Visitor Design Pattern

- main idea and example
- Visitor design code provided for PA4 through PA7
- example reprise using visitor that does traversal
- FAQ about visitors
- Dot visitor
- Other examples including integer and byte expression evaluation

Debugging Ideas for PA3 through PA7

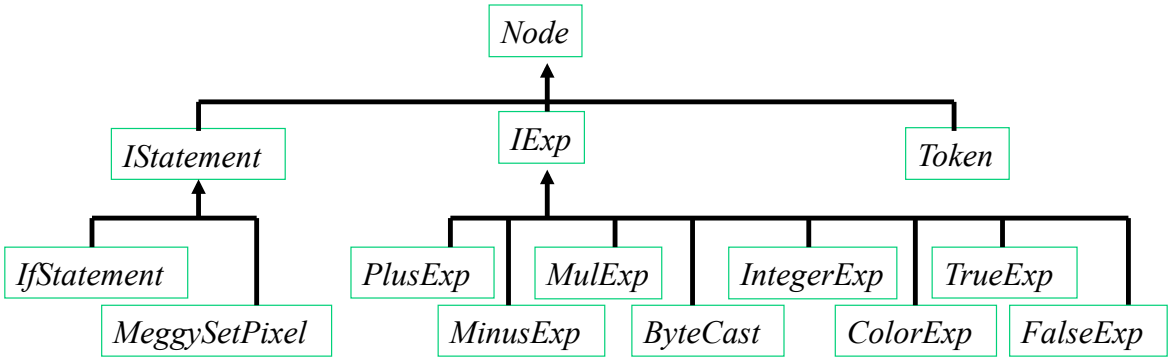
Structure of the MeggyJava Compiler



Grammar Subset and AST Node Hierarchy

```
Statement ::= “if” “(“ Expression “)” Statement “else” Statement
    | “Meggy.setPixel” “(“ Expression “,” Expression “,” Expression “)”

Expression ::=
      Expression ("+" | "-" | "*" ) Expression
    | “(“ “byte” “)” Expression
    | <INTEGER_LITERAL> | <COLOR_LITERAL> | “true” | “false”
```



Line and Position Information

In .lex file

```
"Meggy.Button.B” {return new Symbol(sym.BUTTON_LITERAL,
          new TokenValue(yytext(), yyline+1, yychar));}

{number} {return new Symbol(sym.INT_LITERAL,
          new TokenValue(yytext(), yyline+1, yychar));}
```

In .cup file

```
terminal        TokenValue          INT_LITERAL;

|   BUTTON_LITERAL:n
{: RESULT = new ButtonExp(
                        new Token(n.text,n.line,n.pos)); :}

|   INT_LITERAL:n
{: RESULT = new IntegerExp(
                        new Token(n.text,n.line,n.pos)); :}
```

Syntax-directed Construction of AST

In .cup file

```
non terminal    IExp                exp;
non terminal    List<IStatement>    statement_list;

|    exp:a PLUS exp:b  {: RESULT = new PlusExp(a,b); :}

|    ( E:e )  {: RESULT = e; :}

statement_list ::=
    statement_list:list statement:s
    {: if (s!=null) { list.add(s); }
       RESULT = list; :}

    |   /* epsilon */
    {: RESULT = new LinkedList<IStatement>(); :}

    ;
```


When to use a Visitor Design Pattern

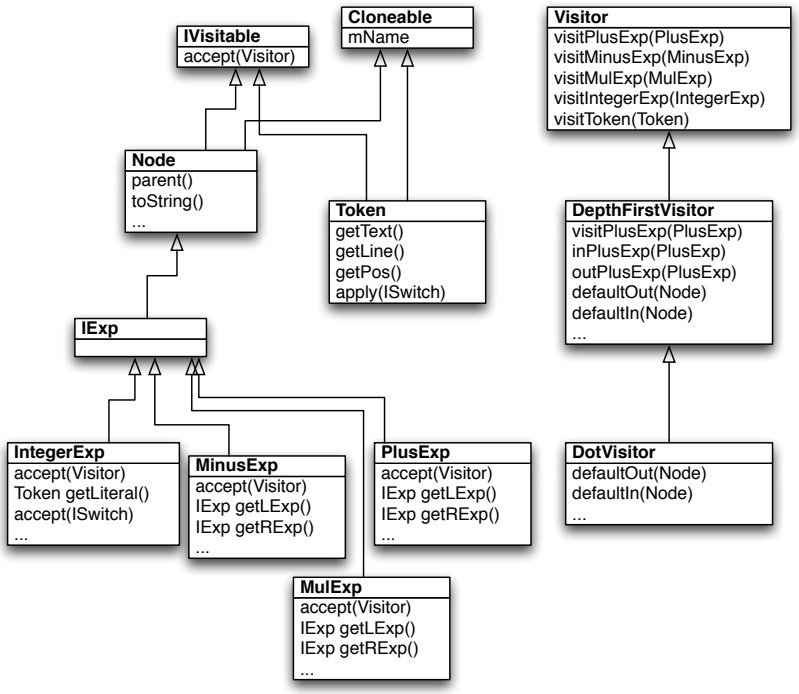
Situation

- Want to perform some processing on all items in a data structure
- Will be adding many different ways to process items, different features
- Will not be changing the classes of the data structure itself much

Possibilities

- For each functionality add a method to all of the classes
  - Each new functionality is spread over multiple files
  - Sometimes can't add methods to existing class hierarchy
- Use a large if-then-else statement in visit method
  - pro: keeps all the code for the feature in one place
  - con: can be costly and involve lots of casting
- Visitor design pattern

Provided ast and visitor class hierarchies



Using the provided visitor design pattern

```
ast_root.accept(new AVRgenVisitor(outfilehandle));
...
 // in class MulExp
public void accept(Visitor v)
{ v.visitMulExp(this); }
...
 // in class DepthFirstVisitor
public void inMulExp(MulExp node) { defaultIn(node); }

public void outMulExp(MulExp node) { defaultOut(node); }

public void visitMulExp(MulExp node)
{
  inMulExp(node);
  if(node.getLExp() != null) { node.getLExp().accept(this); }
  if(node.getRExp() != null) { node.getRExp().accept(this); }
  outMulExp(node);
}
...
```

Snippets from a concrete visitor

```
class AVRgenVisitor extends DepthFirstVisitor {

  HashMap<IExp, Integer> mExpVal = new HashMap<IExp,Integer>();

  public void outPlusExp(PlusExp node) {
    // Look up child values,
    Integer lexpval = mExpVal.get(node.getLExp());
    Integer rexpval = mExpVal.get(node.getRExp());
    // perform operation,
    Integer value =  lexpval + rexpval;
    // and then map computed value to current node.
    mExpVal.put(node, value);
  }

  public void outByteCast(ByteCast node) {
    // Look up child value,
    Integer expval = mExpVal.get(node.getExp());
    // and then map computed value to current node.
    mExpVal.put(node, expval);
  }

}
```


FAQ

How do I associate data with a node in the AST if I can't add fields to the node classes?

What if I want to do the same thing on each node?

What if I only need to do something on certain nodes?

Shouldn't the visit/case methods have return values?

Debugging Ideas

System.out.println in parser actions

Break points in visitor methods

Other debugging ideas will be covered in recitation