Plan for Today

PA1 Peer Review
- Constructive feedback
- Examples of good code
- Highly rated demos
- Suggested evaluation criteria for PA2, will be due Thursday

One pass compilation
- Syntax-Directed, Recursive-Descent, Predictive Parsing and Code Generation

Multi-pass Compilation
- Abstract Syntax Trees (AST)
- Generating code from an abstract syntax tree

Creating an AST in a recursive descent parser

4:30 Review of class so far
- Looking for constructive feedback.

Doing Syntax-Directed Interpretation

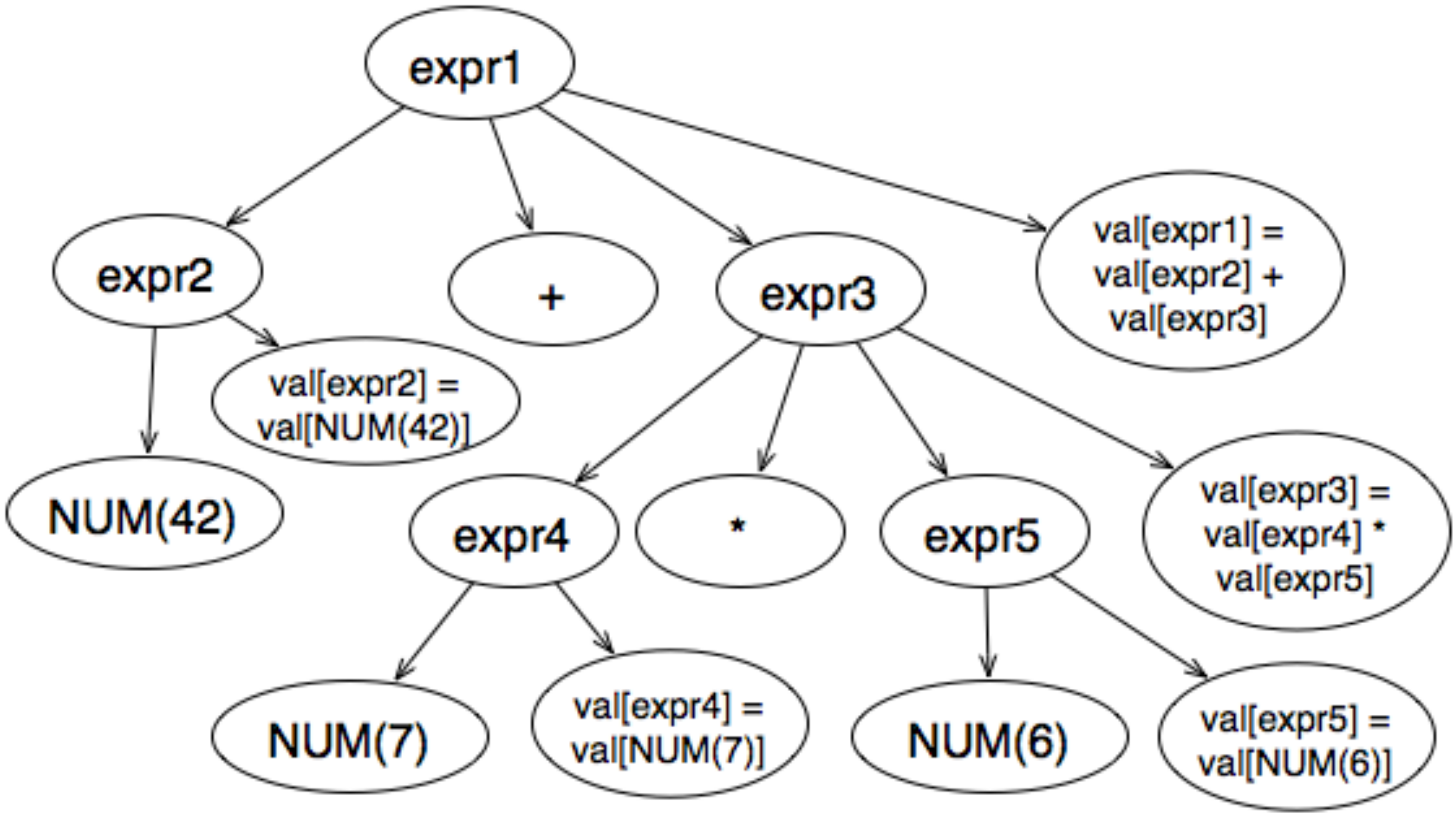
Grammar

(1) exp --> exp * exp
(2) exp --> exp + exp
(3) exp --> NUM

String

42 + 7 * 6

Semantic Rules for Expression Example (Parse Tree w/Actions)



Code Generation versus Interpretation

When interpreting an expression . . .

- Each production matched will result in a computation that generates a value for the expression. Value should be returned.
- Each non terminal on the right hand side of a production has a value associated with it.
- This approach will also be useful when we are building the Abstract Syntax Tree (AST) in PA3, where the value will be the AST we are building.

When did one pass compilation in PA2 . . .

- Each production matched results in a string of target code (in this case AVR assembly)

Example Source and Target Language

Source Language

```
Slist ::= epsilon | S Slist
S     ::= "print" COLOR_LITERAL
```

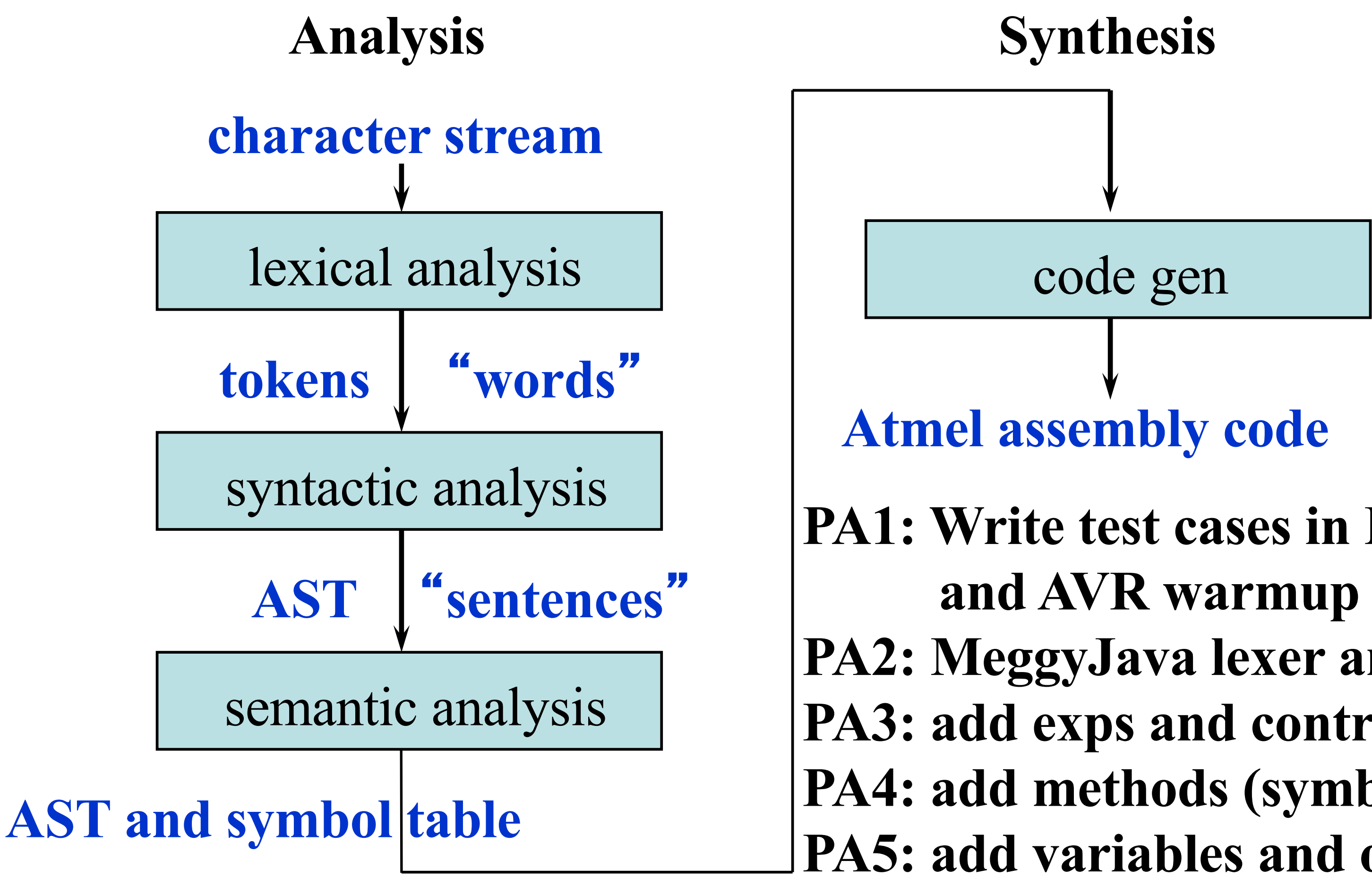
Target Language

- Each print should result in a call to Meggy.setPixel((byte)1,(byte)1, integer for COLOR_LITERAL);
- Essentially the target is a toy subset of the PA2 MeggyJava grammar.

Haskell for …

- Lexer for source language
- Recursive descent predictive parser
- Syntax-directed code generation of the target language

Structure of the MeggyJava Compiler, Multi-pass Compilation



PA1: Write test cases in MeggyJava, and AVR warmup
PA2: MeggyJava lexer and setPixel
PA3: add exps and control flow (AST)
PA4: add methods (symbol table)
PA5: add variables and objects
PA6: add arrays and register allocation

Example program

```
class Byte {
    public static void main(String[] whatever){
            Meggy.setPixel
                (   // Byte multiplication: Byte x Byte -> Int
                    (byte)( (byte)1*(byte)2 ),
                    // Mixed type expression: Byte x Int -> Int
                    (byte)( (byte)3 + 4 ),
                    Meggy.Color.WHITE
                );
    }
}
```

AST of Example Program

How does the AST differ from the parse tree?

Parentheses have been removed

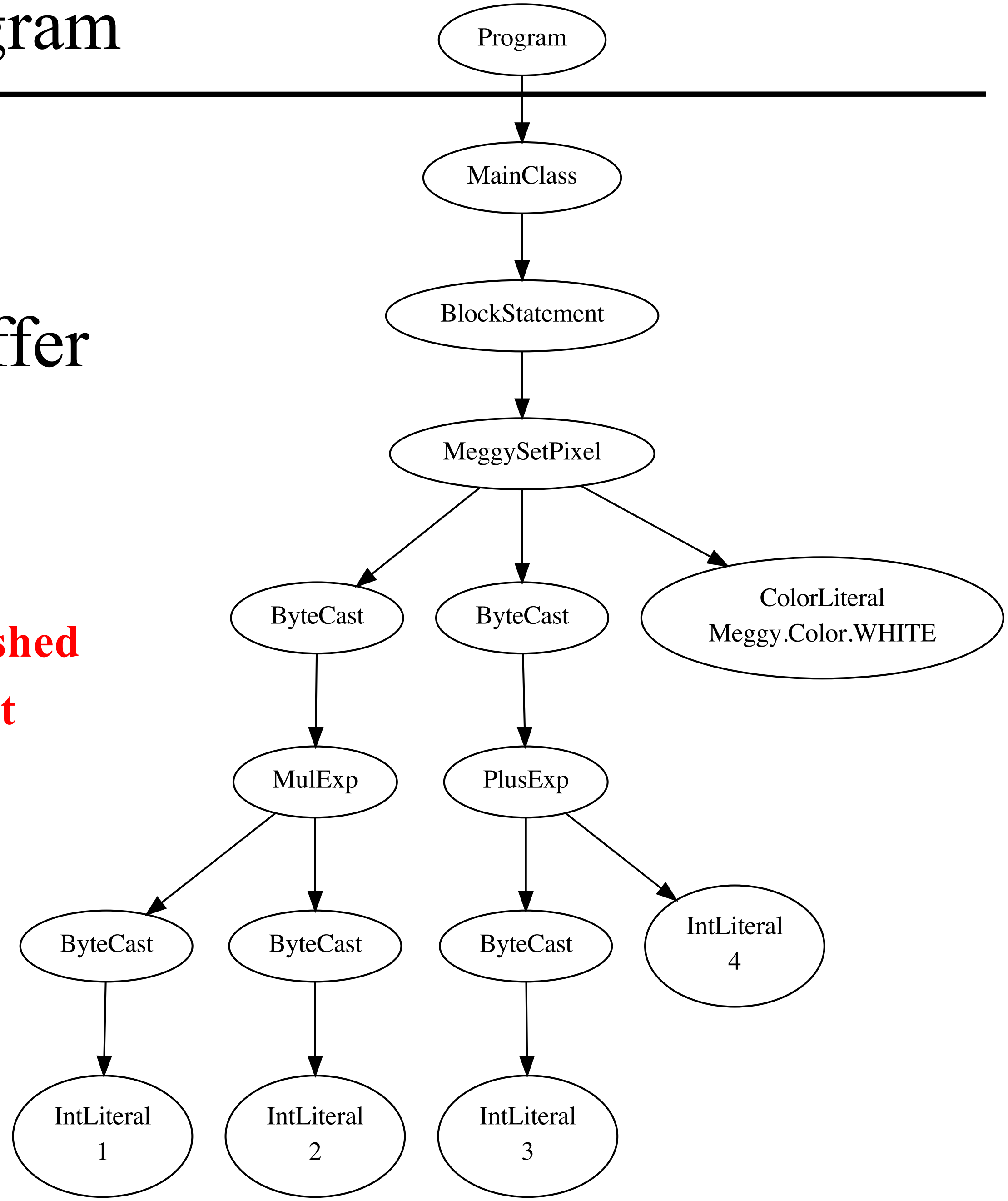
their role -to shape the AST is finished

Some terminals have been pulled out

which?

Some have been pulled up

which?



Code Generation Given an AST

Haskell data type for the AST for example source language

Function that generates code based on that AST

Syntax-directed Construction of AST

Can edit predictive parser to generate ASTs instead of strings.

See example code.

Add in a new statement type