Overview of Approach

PA2

- Syntax-directed code generation

PA3

- Syntax-directed AST creation
- Function for creating the dot file for visualization (TOMORROW)
- Function for checking types (TODAY)
- Function for generating code (TODAY)

Later assignments

- Function for building a symbol table
- Function for allocating memory for variables
- Function for doing register allocation

Type Checking: valid operand types, result types

Signature of an operation (or function)

$inType_1$ **x** $inType_2$ **…** **x** $inType_n$ → outType

How to determine valid inTypes and resulting outType?

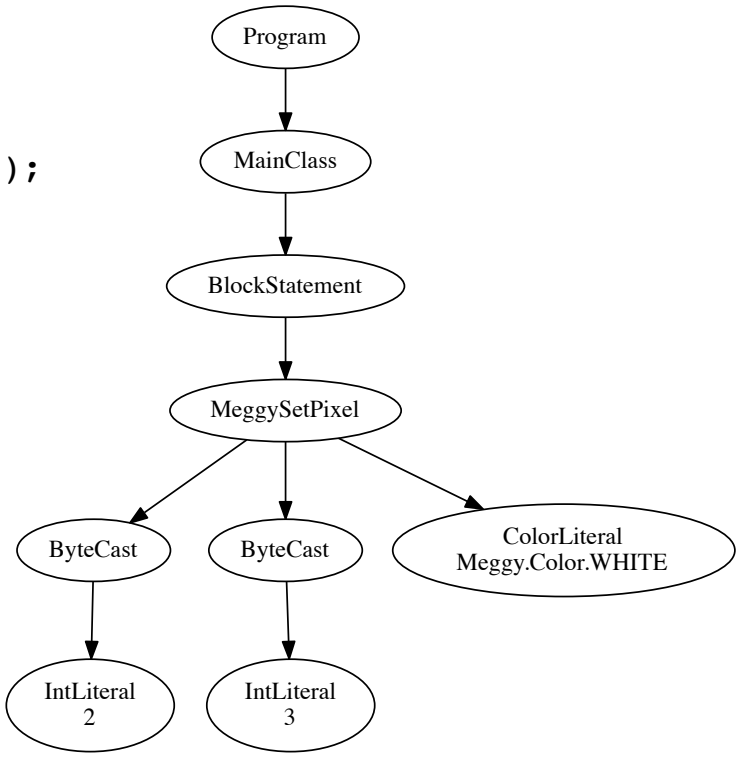
Use given reference compiler MJ.jar and javac.

Create example test programs.

Let's look at an example …

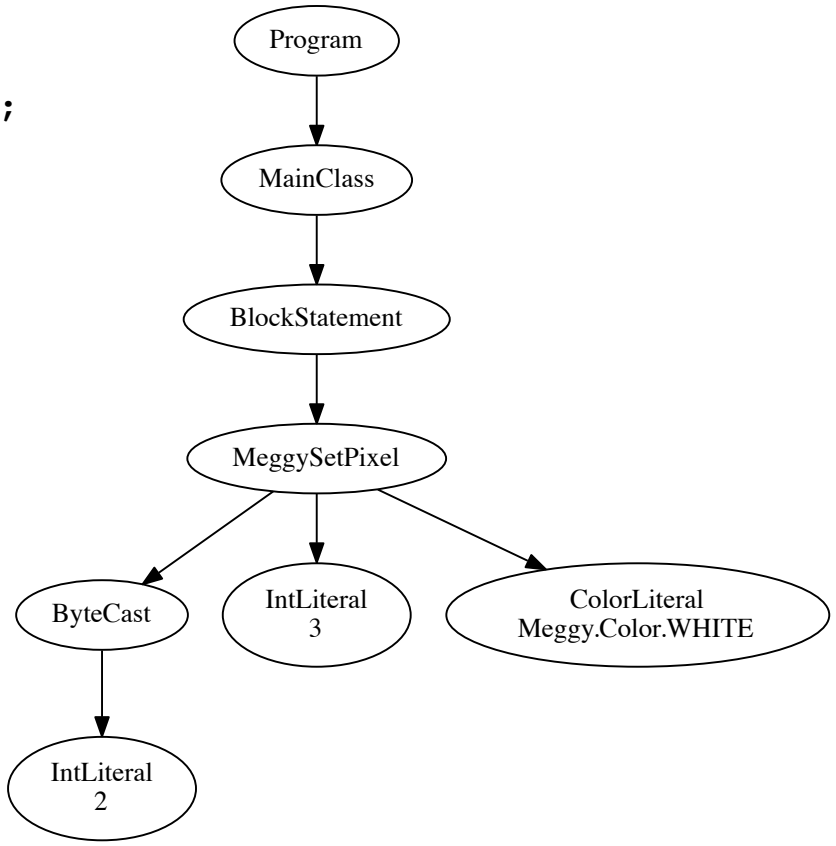
Type checking on an AST (A Correct Input)

```
import meggy.Meggy;

class SetPixelType {
  public static void main(String[] whatever){
    Meggy.setPixel( (byte)2, (byte)3, Meggy.Color.WHITE );
  }
}
```



Type checking on an AST (An Incorrect Input)

```
import meggy.Meggy;

class SetPixelTypeError {
  public static void main(String[] whatever){
    Meggy.setPixel( (byte)2, 3, Meggy.Color.WHITE );
  }
}
```



Traversing a Tree in Haskell

<TreeTraversal.hs code>

- Draw two example trees.
- Add code in preOrder for ThreeKids.
- Write a postOrder function.
- Write an inOrder function.

Implementing Type Checking in Haskell

Will need a user-defined datatype for representing MeggyJava types

- <What are the MeggyJava types for the PA2 grammar?>

Typing function

- Will call typing function on children nodes.
- Will determine if children types are appropriate.
- If there is an error will somehow communicate that error.
- Needs to return type of node if node is an expression node.

Ways to communicate errors in Haskell

- http://www.randomhacks.net/2007/03/10/haskell-8-ways-to-report-errors/
- error "ERROR: what happened"
  - Will stop execution and print an error message.
  - Most decidedly not purely functional.

Strategy: Implement One Language Feature at a Time

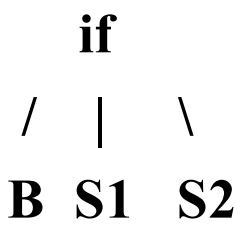
General Strategy: <If Statements will be example we do in class>

- Language features it depends on?
- Go implement them first.
- LEXER: New tokens introduced? How is lexer modified?
- PARSER
  - What new grammar rules are involved?
  - FIRST, FOLLOW, and Nullable?
- AST
  - What new node types will be added to the AST data structure?
  - How do we generate those AST nodes while parsing?
- TYPE CHECKING
  - What are the expected input and output types for the new AST nodes?
  - What are some possible type errors at those nodes?
- CODE GENERATION
  - Assuming operands are on stack, generate code to implement feature.

If Statement code generation

When the visitor encounters ifStmt, simple pre or post order code generation does not suffice. WHY?

We need more complex control:



We need to control the order that code is generated for its children, using branches, jumps and labels.

First, code needs to be generated for the condition (the result of the condition evaluation has been pushed on the RTS) followed by branching instructions, the then block, control to jump over else block, then the else block, and then the end label.

Branches and jumps

An AVR detail: as you know from PA1, conditional branches can only go so far in the code, and code generated, e.g for then or else block is not bounded and thus can exceed that limit. Therefore we have to use jmp sometimes.

Notice: breq is replaced with with a brne followed by a jmp to handle this

```
cp     r24, r25
#WANT breq MJ_L6
brne   MJ_L7
jmp    MJ_L6
MJ_L7:
...  inbounded stretch of code …
MJ_L6:
```

Not Expression: there is no not in AVR, but there is xor

truth table for not and xor

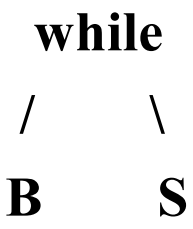
x	y	!x	x xor y
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	0

We can implement NOT x with x XOR 1 :

outNotExp

```
        pop     r24
        ldi     r22, 1
        eor     r24,r22
        push    r24
```

While statement

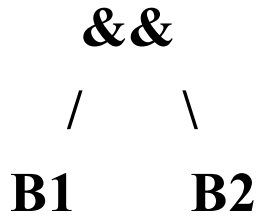


What is the wiring logic?

```
SLbl:
      eval B on stack
      if false jump to endLbL
      gen Code for S
      jump to Slbl
   endLbL:
```

Short circuited (wired) AND, equals

Similar to the If Statement and While Statement, code generation will need to be implemented in the visitAndExp()



can be implemented as: if (B1) return B2 else return false

equalExp, the equality operator ==

Just like in plus and minus, we need to take the mixed type semantics of Java into account, by promoting a byte (1 register) to an int (register pair)