Lexical Structure

<token> ::= <ident> | <int_lit> | <float_lit> | <string_lit> | <reserved> | '&' | '=' | '!' | ',' | '/'
| '==' | '>=' | '>' | '<<' | '<-' | '<=' | '(' | '[' | '<' | '-' | '%' | '!=' | '|' | '+' | '>>' | '->'
| '^' | ')' | ']' | ';' | '*'

<ident> ::= ('a'..'z'|'A'..'Z'|'_'|'$') ('a'..'z'|'A'..'Z'|'$'|'_'|'0'..'9')* but not <reserved>

<int_lit> ::= '0'| (('1'..'9') ('0'..'9')*)

<float_lit> ::= ('0'| ('1'..'9') ('0'..'9')*) '.' ('0'..'9')+

<string_lit> ::= "" ('\' ('b'|'t'|'n'|'f'|'r'|'"'|' ' |'\') | NOT('\'|'"'))* ""
(This is difficult to read. Our language handles escape sequences in string literals the same way as Java)

<reserved> ::= <type> | <image_op> | <color_const> | <boolean_lit> | <other_keyword> |
<color_op>
(Update 2/1: added <color_op> to <reserved>)

<type> ::= 'string' | 'int' | 'float' | 'boolean' | 'color' | 'image' | 'void'

<image_op> ::= 'getWidth' | 'getHeight'

<color_op> ::= 'getRed' | 'getGreen' | 'getBlue'

<color_const> ::= 'BLACK' | 'BLUE' | 'CYAN' | 'DARK_GRAY' | 'GRAY'
| 'GREEN' | 'LIGHT_GRAY' | 'MAGENTA' | 'ORANGE' | 'PINK'
| 'RED' | 'WHITE' | 'YELLOW'

<boolean_lit> ::= 'true' | 'false'

<other_keywords> ::= 'if' | 'else' | 'fi' | 'write' | 'console'

<comment> ::= '#' NOT('\n'|'\r')* ('\r'? '\n')?
(You may assume that a '\r' without a following '\n' will not occur in your input. The rule means that a comment starts with #, and ends with either \r\n, \n, or the end of the input)

<white space> ::= (' '|'\t'|'\r'|'\n')+

<white_space> and <comment> separate tokens, but are otherwise ignored.

Notation

- All elements of the alphabet are surrounded by quotes. Thus '+' is a plus character while + is a metasymbol.
- The only place this doesn't work is in the definition of <string_lit> where ' ' ' indicates a quote character.
- The escape sequences in our language are the same as in Java.

Metasymbols

(,) (parentheses) are used for grouping
| alternative
.. range
* is the Kleene closure: zero or more instances
+ is the positive closure: one or more instances
? means 0 or 1 instance, i.e. a? = a | ε
NOT('\n'|'\r') means any character except newline (\n) and return (\r).

Context Free Grammar

```
Program::=
        (Type  | 'void') IDENT '(' (NameDef ( ',' NameDef)* )? ')'
        (  Declaration ';'     |    Statement ';'  )*                    //yields Program

NameDef::=
        Type IDENT |                          //yields NameDef
        Type Dimension IDENT                  //yields  NameDefWithDimension

Declaration::=
        NameDef (('=' | '<-') Expr)?          //yields  VarDeclaration

Expr::=
   ConditionalExpr | LogicalOrExpr

ConditionalExpr ::=
   'if' '(' Expr ')' Expr 'else'  Expr 'fi'

LogicalOrExpr ::=
   LogicalAndExpr ( '|' LogicalAndExpr)*

LogicalAndExpr ::=
   ComparisonExpr ( '&'  ComparisonExpr)*

ComparisonExpr ::=
   AdditiveExpr ( ('<' | '>' | '==' | '!=' | '<=' | '>=') AdditiveExpr)*

AdditiveExpr ::=
   MultiplicativeExpr ( ('+'|'-') MultiplicativeExpr )*

MultiplicativeExpr ::=
   UnaryExpr (('*'|'/' | '%') UnaryExpr)*

UnaryExpr ::=
       ('!'|'-'| COLOR_OP | IMAGE_OP) UnaryExpr  |
           UnaryExprPostfix

UnaryExprPostfix::=
   PrimaryExpr PixelSelector?

PrimaryExpr ::=
   BOOLEAN_LIT |
   STRING_LIT |
   INT_LIT |
   FLOAT_LIT |
   IDENT |
   '(' Expr ')' |
  ColorConst |                            //yields ColorConstExp
```

```
   '<<' Expr ',' Expr ',' Expr '>>' |              //yields ColorExpr
   'console'                                      //yields ConsoleExpr

PixelSelector::=
    '[' Expr ',' Expr ']'

Dimension::=
          '[' Expr ',' Expr ']'                   //yields Dimension


Statement::=
          IDENT PixelSelector? '=' Expr |         //yields AssignmentStatement
          IDENT PixelSelector? '<-' Expr |        //yields  ReadStatement
          'write' Expr '->' Expr |                //yields WriteStatement
          '^' Expr                                //yields ReturnStatement
```

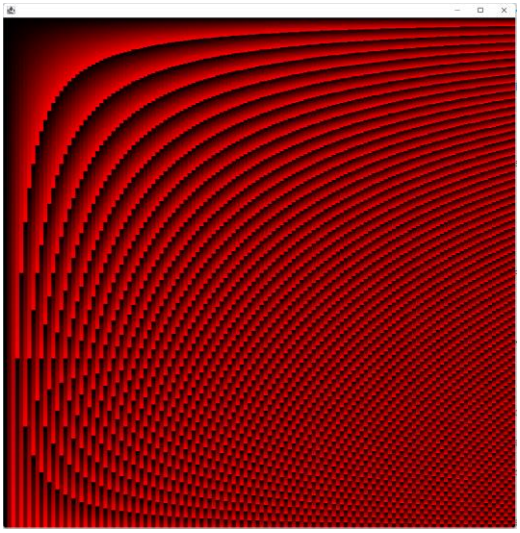
Type System

Scope Rules

1. With one exception, all variables are global and must be declared before they are used.
2. ASTNodes subclasses that reference variables should be annotated with the corresponding Declaration. The relevant types are ReadStatement, AssignmentStatement, and IdentExpr.
3. The exception to the rule that all variables are global are the expressions appearing in PixelSelector that appear on the left side of an assignment statement are local variables defined in the assignment statement. These variables are implicitly declared to have type INT, and must be an IdentExpr. The names cannot be previously declared as global variable.

Example:

```
image BDP0(int size)
 int Z = 255;
 image[size,size] a;
 a[x,y] = <<(x/8*y/8)%(Z+1), 0, 0>>;
 ^ a;
```



a. The above program, which generates and returns the above image (size = 1024, but reduced for this document) explicitly declares two global variables Z and a. Input parameter size is also treated as a global variable.

b. Variables x and y are local variables that are implicitly declared and valid only in the scope of the assignment statement. Names x and y are not previously declared. A Declaration must be created for these names and added to the symbol table before processing the right hand side of the assignment statement, and removed from the symbol table after processing the right hand side is finished.

1. Your type checker will statically ensure that all global variables are initialized before they are used. The easiest way to do this is to add a boolean field with appropriate getters and setters to the Declaration class. In the above example, Z is initialized in its declaration. Variable a is not initialized in its declaration but is initialized in the assignment statement. When referencing variables (right side of assignment for Z, and the return statement (^a;) for a, check that they are initialized.
2. Input parameters are implicitly initialized. Thus, size would be marked as initialized.
3. The name of the program cannot be reused. In the example, this is BDP0.
4. See additional rules for Declarations below
5. Your symbol table should use an appropriate data structure.

Expressions

1. A type is inferred for all Expressions. This is saved in the Type field.
2. Hint: it is convenient to let your visit methods for expressions return the type.
3. Depending on the context, some types may be coerced into other types. The type to be coerced to is saved in a separate field. (Many expressions are not coerced. Either setting the coerced type to the inferred type, or leaving it null will work)
4. Literals

ASTNode type	**Inferred Type (value in Types.Type)**
BooleanLitExpr	BOOLEAN
StringLitExpr	STRING
IntLitExpr	INT
FloatLitExpr	FLOAT
ColorConstExpr	COLOR
ConsoleExpr	CONSOLE

1. ColorExpr

. All component types are INT or all component types are FLOAT

a. If INT, inferred expression type is COLOR, otherwise COLORFLOAT, which has been added to Types.Type enum.

2. UnaryExpr

. The following combinations of types and operators are allowed with indicated inferred type.
a.

Operator	**Expr type**	**Inferred type**
!	BOOLEAN	BOOLEAN
-	INT	INT
-	FLOAT	FLOAT
COLOR_OP (getRed,etc.)	INT	INT
COLOR_OP	COLOR	INT
COLOR_OP	IMAGE	IMAGE
IMAGE_OP (getWidth, getHeight)	IMAGE	INT

1. IdentExpr

. Lookup name in symbol table, it must be declared.
a. Type is obtained from the declaration in the symbol table.

1. ConditionalExpr

. Type of condition must be BOOLEAN
a. Type of trueCase must be the same as the type of falseCase
b. Type is the type of trueCase

2. UnaryPostfix

. Type of expr must be Image
a. Types of expressions in the PixelSelector must be INT
b. Type is INT

3. BinaryExpr

OPERATOR	**left.Type**	**right.Type**	**Inferred type**
AND,OR	BOOLEAN	BOOLEAN	BOOLEAN
EQUALS, NOT_EQUALS	leftType == RightType		BOOLEAN
PLUS, MINUS	INT	INT	INT
	FLOAT	FLOAT	FLOAT
	INT, coerce to FLOAT	FLOAT	FLOAT

	FLOAT	INT, coerce to FLOAT	FLOAT
	COLOR	COLOR	COLOR
	COLORFLOAT	COLORFLOAT	COLORFLOAT
	COLORFLOAT	COLOR, coerce to COLORFLOAT	COLORFLOAT
	COLOR, coerce to COLORfLOAT	COLORFLOAT	COLORFLOAT
	IMAGE	IMAGE	IMAGE
TIMES,DIV,MOD	All cases for PLUS,MINUS		
	IMAGE	INT	IMAGE
	IMAGE	FLOAT	IMAGE
	INT, coerce to COLOR	COLOR	COLOR
	COLOR	INT, coerce to COLOR	COLOR
	FLOAT, coerce to COLORFLOAT	COLOR, coerce to COLORFLOAT	COLORFLOAT
	COLOR,, coerce to COLORFLOAT	FLOAT, coerce to COLORFLOAT	COLORFLOAT
LT,LE,GT,GE	INT	INT	BOOLEAN
	FLOAT	FLOAT	BOOLEAN
	INT, coerce to FLOAT	FLOAT	BOOLEAN
	FLOAT	INT, coerce to FLOAT	BOOLEAN

Statements

1. WriteStatement
. Type of Dest must be STRING or CONSOLE
a. Type of Source cannot be CONSOLE
2. ReadStatement
. Get target type by looking up lhs var name in symbol table.
a. A read statement cannot have a PixelSelector
b. The right hand side type must be CONSOLE or STRING
c. Mark target variable as initialized.
3. AssignmentStatement

. Get target type by looking up lhs var name in symbol table. Save type of target variable and its Declaration.

a. Target variable is marked as initialized.

b. CASE: target type is not IMAGE

.There is no PixelSelector on left side.

i.Expression must be assignment compatible with target.

0. If the expression type and target variable type are the same, they are assignment compatible.
1. The following pairs are assignment compatible. The expression is coerced to match the target variable type.

Variable	Expression
INT	FLOAT
FLOAT	INT
INT	COLOR
COLOR	INT

a. CASE: target type is an IMAGE without a PixelSelector

.Expression must be assignment compatible with target

i.If both the expression and target are IMAGE, they are assignment compatible

ii.The following pairs are assignment compatible. If indicated, the expression should be coerced to the indicated type.

Variable	Expression
IMAGE	INT, coerce to COLOR
IMAGE	FLOAT, coerce to COLORFLOAT
IMAGE	COLOR
IMAGE	COLORFLOAT

a. CASE: target type is an IMAGE with a PixelSelector

.Recall from scope rule: expressions appearing in PixelSelector that appear on the left side of an assignment statement are local variables defined in the assignment statement. These variables are implicitly declared to have type INT, and must be an IdentExpr. The names cannot be previously declared as global variable.

i.Type of right hand side must be COLOR, COLORFLOAT, FLOAT, or INT, and is coerced to COLOR.

b. Note that COLORFLOAT objects are only created implicitly. There is no way to declare a variable of type COLORFLOAT

1. Return statement

. Declared type of program must be the same as the type of expression.

Declarations

1. Visit NameDef (or NameDefWithDim) to insert name in symbol table. See discussion of scope rules above.
2. If type of variable is Image, it must either have an initializer expression of type IMAGE, or a Dimension.
3. For Dimensions, both expressions must have type INT
4. If VarDeclaration has an assignment initializer, the right hand side type must be assignment compatible as defined above for Assignment Statements.
5. If VarDeclaration has a read initializer, the right hand side type must be assignment compatible as defined above for Read Statements.

Program

1. Handle parameters as NameDef (not NameDefWithDim). Mark name as initialized.
2. Visit nodes in decsAndStatements.

Code Generation:

In assignment 5, we will partially implement code generation for part of our language. We will not handle any programs with color or images. This means that your visitor does not need to implement visit methods for NameDefWithDim, Dimension, ColorConst, UnaryExprPostFix, PixelSelector. In addition, we will not do IO from files or URLs: we will just read and write strings, ints, booleans, and floats from the console.

The rest of the language will be implemented in Assignment 6.

ASTNode	Template for Java code.

Program	<package declaration>
	<imports>
	public class <name> {
	  public static <returnType> apply(<params>){
	  <decsAndStatements>
	  }
	}

<params>	Comma separated list of NameDef
<imports>	List of import statements for any classes from package edu.ufl.cise.plc.runtime that are invoked in generated code.
<decsAndStatements>	list of Declarations and Statements
NameDef	<type> <name>
VarDeclaration (only read initializers from console for assignment 5)	<nameDef> ; Or if this has an assignment or read initializer <nameDef> = <expr>
ConditionalExpr	(<condition>) ? <trueCase> : <falseCase>
BinaryExpr	(<left> <op> <right>)
BooleanLitExpr	Java literal corresponding to value (i.e. true or false)
ConsoleExpr	(<boxed(coerceTo)> ConsoleIO.readValueFromConsole(“coerceType”, <prompt>) <prompt> is a string that requests the user to enter the desired type. <boxed(type)> means the object version of the indicated type: Integer, Boolean, Float, etc.

	The first argument of readValueFromConsole is an all uppercase String literal with corresponding to the type. (i.e. one of “INT”, “STRING”, “BOOLEAN”, “FLOAT”) For example, if the PLCLang source has j <- console; where j is int, then this would translate to j = (Integer) ConsoleIO.readValueFromConsole(“INT”, “Enter integer:”); Note that the “j = “ part would be generated by the parent AssignmentStatement. See the provided ConsoleIO class.
FloatLitExpr	Java float literal corresponding to value. If coerceTo != null and coerceTo != FLOAT, add cast to coerced type. Recall Java float literals must have f appended. E.g. 12.3 in source is 12.3f in Java. (12.3 in Java is a double–if you do this your program will probably run, but fail test cases that check for equality)
IntLitExpr	Java int literal corresponding to value If coerceTo != null and coerceTo != INT, add cast to coerced type.
IdentExpr	<identExpr.getText> If coerceTo != null and coerceTo != identExpr.type, add cast to coerced type.
StringLitExpr	“”” <stringLitExpr.getValue>””” (we will not handle escape sequences in String literals in this assignment)
UnaryExpr (for assignment 5, only - and !)	(<op> <expr>)
ReadStatement (only read from console in assignment 5)	<name> = <consoleExpr> ;
WriteStatement	ConsoleIO.console.println(<source>) ;

(only write to console in assignment 5)	println here is just the usual PrintStream method. Usually this is used with the PrintStream instance System.out. For this assignment, you should instead use the PrintStream object ConsoleIO.console. This will typically be assigned to System.out, but may be changed for grading or other purposes.
AssignmentStatement	<name> = <expr> ;
ReturnStatement	return <expr> ;

In the above <...> is used to indicate that something is filled in, usually by visiting the corresponding AST node.

In the CodeGenVisitor, each visit method should append the corresponding Java code to a StringBuilder object. It is convenient to pass in the StringBuilder as an argument to the visitMethod and return a StringBuilder as a result.

I found it convenient to implement a new class that delegates to StringBuilder and provides methods like comma(), semi(), lparen(), etc. that append the corresponding character. These methods should always return this to allow chaining: sb.append(name).comma().append(name);

Your generated code will be more readable if \n are added in appropriate places, but the format of your generated code will not be graded.

In assignment 6, we will implement code generation for the rest of our language. It will be helpful to review the lecture slides (and maybe the lecture itself) from 323, 3/25. Note that the slides have been updated so that the examples match the provided code.

ASTNode	Template for Java code.

Program	<package declaration>
	<imports>
	public class <name> {
	  public static <returnType> apply(<params>){
	  <decsAndStatements>
	  }
	}

<params>	Comma separated list of NameDef
<imports>	List of import statements for any classes from package edu.ufl.cise.plc.runtime that are invoked in generated code.
<decsAndStatements>	list of Declarations and Statements
NameDef	<type> <name>

<table>
<tr><td>VarDeclaration</td><td><nameDef> ;

Or if this has an assignment or read initializer

<nameDef> = <expr> or <nameDef> <- <expr>

Handle as described in AssignmentStatement or ReadStatement

If the PLCLang type is image, implement with a java.awt.image.BufferedImage.

<table><tr><td></td><td>Has dimension</td><td>Does not have dimension</td></tr><tr><td>Has initializer</td><td>Read image using readImage(String,int,int) method in FileURLIO</td><td>Read image using readImage(String) method in FileURLIO</td></tr><tr><td>Does not have expr</td><td>Instantiate a new BufferedImage of the given dimension and type BufferedImage.TYPE_INT_RGB</td><td>This case should have been marked as error during type checking.</td></tr></table>
If the PLCLang type is color, implement with edu.ufl.cise.plc.runtime.ColorTuple.</td></tr>
<tr><td>ConditionalExpr</td><td>(<condition>) ? <trueCase> : <falseCase></td></tr>
<tr><td>BinaryExpr</td><td>(<left> <op> <right>)

BinaryExpressions on color objects are performed componentwise. There are routines in edu.ufl.cise.plc.runtime.ImageOps that might be useful.

BinaryExpressions on image objects are performed pixelwise. There are routines in edu.ufl.cise.plc.runtime.ImageOps that might be useful.

If a binary operation has one image and a color, apply the operation between pixel and color to all operations in the image. If a binary operation involves an image an int, create a color with all color components equal to the int value and then apply pixelwise to the image. In other words for image im0 and int k =, im0 * k = im0 * <<k,k,k>>. (There is a ColorTuple constructor for this case)</td></tr>
<tr><td>BooleanLitExpr</td><td>Java literal corresponding to value (i.e. true or false)</td></tr>
</table>

ConsoleExpr	(<boxed(coerceTo)> ConsoleIO.readValueFromConsole(“coerceType”, <prompt>) <prompt> is a string that requests the user to enter the desired type. <boxed(type)> means the object version of the indicated type: Integer, Boolean, Float, etc. The first argument of readValueFromConsole is an all uppercase String literal with corresponding to the type. (i.e. one of “INT”, “STRING”, “BOOLEAN”, “FLOAT”) For example, if the PLCLang source has j <- console; where j is int, then this would translate to j = (Integer) ConsoleIO.readValueFromConsole(“INT”, “Enter integer:”); Note that the “j = “ part would be generated by the parent AssignmentStatement. See the provided ConsoleIO class. Color types are read from console similarly to the other types–the user inputs three values for the three color components.
FloatLitExpr	Java float literal corresponding to value. If coerceTo != null and coerceTo != FLOAT, add cast to coerced type. Recall Java float literals must have f appended. E.g. 12.3 in source is 12.3f in Java. (12.3 in Java is a double–if you do this your program will probably run, but fail test cases that check for equality)
IntLitExpr	Java int literal corresponding to value If coerceTo != null and coerceTo != INT, add cast to coerced type.
IdentExpr	<identExpr.getText> If coerceTo != null and coerceTo != identExpr.type, add cast to coerced type.
StringLitExpr	“”” <stringLitExpr.getValue>””” (we will not handle escape sequences in String literals in this assignment)

UnaryExpr	(<op> <expr>) If op is getRed, getGreen, or getBlue If expr is int, it is interpreted as packed pixel. Use routine in ColorTuple class to get color value. If expr is color, return color component. For convenience, ColorTuple has overloaded methods that take either an int or ColorTuple, so these types may be handled uniformly during Code Generation If expr is image, use extractRed, etc. routines in ImageOps
ReadStatement	<name> = <consoleExpr> ; Use routines in FileURLIO to handle reading from a file or URL. If the target type is image, then the value read will be a string interpreted as url of filename.
WriteStatement	ConsoleIO.console.println(<source>) ; println here is just the usual PrintStream method. Usually this is used with the PrintStream instance System.out. For this assignment, you should instead use the PrintStream object ConsoleIO.console. This will typically be assigned to System.out, but may be changed for grading or other purposes. If type is image and target is console, use displayImageOnScreen method in ConsoleIO. If target is a file, use writeImage in FileURLIO for image types and writeValue for other types.
AssignmentStatement	<name> = <expr> ; If <name>.type is image and <expr>.type is image, there are two cases, depending on whether the <name> was declared with a Dimension or not. If declared with a Dimension, the image <name> always keeps the declared size. The assignment is implemented by evaluating the right hand size and calling ImageOps.resize. If not declared with a size, the image <name> takes the size of the right hand side image. If <expr> is an identExpr, the rhs image is cloned using ImageOps.clone If <expr>.coerceTo is color, the color is assigned to every pixel in the image. If <expr>.coerceTo is int, the int is used as a single color component in a ColorTuple where all three color components have the value of

	the int. (The value is truncated, so values outside of [0, 256) will be either white or black.)
ReturnStatement	return <expr> ;
ColorConstExpr	Interpret the color constants as predefined instances of the java.awt.Color class. Use getRGB routine to get a packed pixel, unpack it, and create a ColorTuple object.
ColorExpr	Generate code to evaluate each color component expression and create a ColorTuple object.
Dimension	Usually, the width and height are used as parameters, for example to the BufferedImage constructor. It is convenient for the visit method to generate code to evaluate width expression, comma, code to evaluate height expression.
PixelSelector	Handled differently depending on whether they are on the left or right side of expression. If on the left side, they are the index variables of a nested for loop. If they are on the right side, they will be used as parameters and generate code to evaluate X, comma, code to evaluate Y.
UnaryExprPostfix	The expression will be something like a[e0,e1] where a is an image and [e0,e1] is represented by a PixelSelector. Invoke the BufferedImage getRGB method with the expressions in the PixelSelector as parameters and unpack the returned int to create a ColorTuple

Provided code in edu.ful.cise.plc.runtime.jar
edu.ufl.plc.runtime

- ColorTuple
  - class used to represent the color type
  - Contains useful methods for working with colors, including conversion to and from packed ints.
- ColorTupleFload
  - class used to represent the color type when the components are float.
- FileURLIO
  - class supporting reading and writing of images and other types to and from files
  - Reading images from URLs is also supported.
- PLCRuntimeException
  - A subclass of RuntimeException (so does not need to be declared in throws clauses) that is thrown by routines in the edu.ufl.plc.runtime package

edu.ufl.plc.runtime.javaCompilerClassLoader

- PLCLangExec

    - Provides an exec method. This method is passed a String containing PLCLang code which is dynamically compiled and executed.
- DynamicClassLoader
- DynamicCompiler, which uses the following
    - InMemoryByteCodeObject
    - InMemoryClassFileManager
    - StringJavaFileObject

Also provided

Assignment6TestStarter

Other comments

1. Make sure your code works if the package name is an empty string. In this case, the generated class belongs to the default class and there is not package declaration.
2. Do not change any of the provided classes in the edu.ufl.cise.plc.runtime or edu.ufl.cise.plc.runtime.javaCompilerClassLoader packages. You may add additional classes to the edu.ufl.cise.plc.runtime package.