

Assignment 4

Implement a class `TypeChecker` that implements the `ASTVisitorInterface` and performs type checking as specified in the following attribute grammar/action routine. A symbol table implementation will also be needed.

Abstract Syntax	Semantic rules and conditions
Program ::= IDENTIFIER Block	
Block ::= enterScope (Declaration Statement)*	Block ::= <code>enterScope</code> (Declaration Statement)* <code>leaveScope</code>
Declaration ::= Type IDENTIFIER (ϵ Expression ₀ Expression ₁)	Declaration.name \leftarrow IDENTIFIER.name Declaration.name \notin SymbolTable.currentScope Expression ₀ == ϵ or (Expression ₀ .type == integer and type == image) Expression ₁ == ϵ or Expression ₁ .type == integer and type == image (Expression ₀ == ϵ) == (Expression ₁ == ϵ) SymbolTable \leftarrow SymbolTable \cup (name, Declaration)
Type ::= int float boolean image filename	
Statement ::= StatementInput StatementWrite StatementAssign StatementWhile StatementIf StatementShow StatementSleep	
StatementInput ::= IDENTIFIER Expression FYI: The value of the expression indicates the index of the input in the array of command line parameters, so it needs to be an integer.	StatementInput.destName \leftarrow IDENTIFIER.name StatementInput.dec \leftarrow SymbolTable.lookup(StatementInput.destName) StatementInput.dec != null Expression.type == integer
StatementWrite ::= IDENTIFIER ₀ IDENTIFIER ₁	StatementWrite.sourceName \leftarrow IDENTIFIER ₀ .name StatementWrite.sourceDec \leftarrow symbolTable.lookup(StatementWrite.sourceName) StatementWrite.sourceDec != null StatementWrite.destName \leftarrow IDENTIFIER ₁ .name StatementWrite.destDec \leftarrow symbolTable.lookup(StatementWrite.destName) StatementWrite.destDec != null sourceDec.type == image

	destDec.type == filename
StatementAssign ::= LHS Expression	LHS.type == Expression.type
StatementWhile ::= Expression Block	Expression.type == boolean
StatementIf ::= Expression Block	Expression.type == boolean
StatementShow ::= Expression	Expression.type \in {int, boolean, float, image}
StatementSleep ::= Expression	Expression.type == integer
LHSIdent ::= IDENTIFIER	LHSIdent.name \leftarrow IDENTIFIER.name LHSIdent.dec \leftarrow SymbolTable.lookup(LHSIdent.name) LHSIdent.dec != null LHSIdent.type \leftarrow LHSIdent.dec.type
LHSPixel ::= IDENTIFIER PixelSelector	LHSPixel.name \leftarrow IDENTIFIER.name LHSPixel.dec \leftarrow SymbolTable.lookup(LHSPixel.name) LHSPixel.dec != null LHSPixel.dec.type == image LHSPixel.type \leftarrow integer
LHSSample ::= IDENTIFIER PixelSelector Color	LHSSample.name \leftarrow IDENTIFIER.name LHSSample.dec \leftarrow SymbolTable.lookup(LHSSample.name) LHSSample.dec != null LHSSample.dec.type == image LHSSample.type \leftarrow integer
Color ::= red green blue alpha	
PixelSelector ::= Expression ₀ Expression ₁	Expression ₀ .type == Expression ₁ .type Expression ₀ .type == integer or Expression ₀ .type == float
Expression ::= ExpressionBinary ExpressionConditional ExpressionFunctionAppWithExpressionArg ExpressionFunctionAppWithPixelArg ExpressionPixel ExpressionPixelConstructor ExpressionPredefinedName ExpressionUnary ExpressionIdent ExpressionIntegerLiteral ExpressionBooleanLiteral ExpressionFloatLiteral	Expression.type \leftarrow type of right hand side expression
ExpressionConditional ::= Expression ₀ Expression ₁ Expression ₂	Expression ₀ .type == boolean Expression ₁ .type == Expression ₂ .type ExpressionConditional.type == Expression ₁ .type ExpressionConditional.type \leftarrow Expression ₁ .type
ExpressionBinary ::= Expression ₀ op Expression ₁	ExpressionBinary.type \leftarrow inferredType(Expression0.type, Expression1.type, op) (inferredType is defined below)
ExpressionUnary ::= Op Expression	ExpressionUnary.type \leftarrow Expression.type
ExpressionIdent	ExpressionIdent.dec \leftarrow SymbolTable.lookup(ExpressionIdent.name) ExpressionIdent.dec != null ExpressionIdent.type \leftarrow ExpressionIdent.dec.type

ExpressionIntegerLiteral	ExpressionIntegerLiteral.type \leftarrow integer
ExpressionBooleanLiteral	ExpressionBooleanLiteral.type \leftarrow boolean
ExpressionFloatLiteral	ExpressionFloatLiteral.type \leftarrow float
ExpressionPixelConstructor ::= Expression _{alpha} Expression _{red} Expression _{green} Expression _{blue}	Expression _{alpha} .type == integer Expression _{red} .type == integer Expression _{green} .type == integer Expression _{blue} .type == integer Expression.type \leftarrow integer ExpressionPixelConstructor.type \leftarrow integer
ExpressionPixel ::= IDENTIFIER PixelSelector	ExpressionPixel.name \leftarrow IDENTIFIER.name ExpressionPixel.dec \leftarrow SymbolTable.lookup(ExpressionPixel.name) ExpressionPixel.dec != null ExpressionPixel.dec.type == image ExpressionPixel.type \leftarrow integer
ExpressionFunctionAppWithExpressionArg ::= FunctionName Expression	ExpressionFunctionAppWithExpressionArg.type \leftarrow inferredTypeFunctionApp(FunctionName, Expression.type) (see below)
ExpressionFunctionAppWithPixel ::= FunctionName Expression ₀ Expression ₁	if (FunctionName == cart_x FunctionName == cart_y) Expression ₀ .type == float Expression ₁ .type == float ExpressionFunctionAppWithPixel \leftarrow integer if (FunctionName == polar_a FunctionName == polar_r) Expression ₀ .type == integer Expression ₁ .type == integer ExpressionFunctionAppWithPixel \leftarrow float
ExpressionPredefinedName	ExpressionPredefinedName.type \leftarrow integer
FunctionName ::= sin cos atan abs log cart_x cart_y polar_a polar_r int float width height Color	

This table gives the legal argument types for operators and functions along with the inferred type, which is the type of the result. If you are confronted with a combination not in the table, it is not legal.

Expression ₀ .type	Expression ₁ .type	Operator	inferred type for ExpressionBinary.type
integer	integer	+, -, *, /, %, **, &,	integer

float	float	+, -, *, /, **	float
float	integer	+, -, *, /, **	float
integer	float	+, -, *, /, **	float
boolean	boolean	&,	boolean
integer	integer	&,	integer
integer	integer	==, !=, >, >=, <, <=	boolean
float	float	==, !=, >, >=, <, <=	boolean
boolean	boolean	==, !=, >, >=, <, <=	boolean
Expression.type		Function	inferred type for ExpressionFunctionAp pWithExpressionArg
integer		abs, red, green, blue, alpha	integer
float		abs, sin, cos, atan, log	float
image		width, height	integer
int		float	float
float		float	float
float		int	int
int		int	int

- TypeChecker.java, TypeCheckerTest.java, and Types.java have been provided. You will need to complete the implementations of TypeChecker.java and of course, add more tests to TypeCheckerTest.java.
- You will also need to implement a data structure for your symbol table. An implementation of the Leblanc-Cook symbol table that was discussed in class is recommended. The specification above assumes that your symbol table has a method lookup that will return a Declaration if an identifier has been declared and is visible in the current scope. Otherwise, it will return null.
- Some of the AST nodes are already decorated with attribute values (name, destName, value, etc) that were obtained from the Scanner when the AST was constructed (Assignment 3). In this assignment, type and dec attributes need to be added for some nodes. If an attribute is a type,

its declared type should be a value from the enum `Types.Type`. A `dec` attribute should be a Declaration.

- `TypeCheckerTest.java`, provides a few Junit tests to illustrate how the pieces fit together. Currently, all three tests fail due to an `UnsupportedOperationException`. All tests should pass once you are finished.
- The provided class `Types` contains an enum `Type`. Do not change the names in the enum or reorder them. You should not need to modify `Types.java` for this assignment.
- If a type error is discovered, throw a `SemanticException`. The `Token` argument should be the first `Token` of the AST node where the error was detected. As in previous assignments, the contents of error messages will not be graded, but you will be much happier in future assignments if they are descriptive and helpful.
- Wherever possible, fields to represent attributes should be declared in abstract classes so they will be inherited by all subclasses and can be accessed without needing a cast. (For example, put the type attribute in `Expression` where it will be inherited by all the concrete expression classes.) It is often convenient to return attributes from the visit methods where they were computed. This is especially the case for the type of expressions.
- In the specification, for convenience, `symbolTable` is treated as a global attribute rather than being redefined everywhere as an inherited attribute. This can be directly implemented by making a reference to `symbolTable` a field in your `ASTVisitor`. You will need to design and implement an appropriate data structure.

Turn in a jar file containing your source code for `TypeChecker.java`, `TypeCheckerTest.java`, `Parser.java`, `Scanner.java`, all of the AST classes, `Types.java`, and any classes you may have added. Make sure your symbol table is included.

Your `TypeCheckerTest` will not be graded, but may be looked at in case of academic honesty issues. We will subject your submission to our set of unit tests and your grade will be determined solely by how many tests are passed.

Name your jar file in the following format: *firstname_lastname_ufid_hw4.jar*

Comments and Suggestions

- Remember that when you submit your assignment, you are attesting that have neither given nor received inappropriate help on the assignment. In this course, all assignments must be your own individual work, including the Scanner and Parser after they have been graded.
- As in previous assignments, work incrementally. It is useful to throw an `UnsupportedOperationException` in visit routines that have not been implemented yet rather than returning null. The provided version of `TypeChecker.java` has done this for you. Once your implementation is completed, traces of this exception should be eliminated.
- Review the lecture on the Visitor Pattern before you begin.

- To get more out of the project, as you implement it, think about which attributes are synthesized and which are inherited. Would it be possible to incorporate this type checking with parsing?