Assignment 5

Due: Fri. Nov 17 at 11:59pm

Implement a CodeGenVisitor class to traverse the decorated AST and generate code. The abstract syntax has been annotated to indicate which part of the language is to be implemented in Assignment 5 and how it maps into JVM elements. The rest of the language will be implemented in Assignment 6.

Program ::= name (Declaration | Statement)*

Generate code for a class called name.

statements are added to main method

Declaration ::= Declaration_Image | Declaration_SourceSink | Declaration_Variable

Declaration_Image ::= name (xSize ySize | ε) Source

TODO Assignment 6

Declaration_SourceSink ::= Type name Source

TODO Assignment 6

Declaration_Variable ::= Type name (Expression | ε)

Add field, name, as static member of class.
If there is an expression, generate code to evaluate it and store the results in the field.
See comment about this below.

Statement ::= Statement_Assign | Statement_In | Statement_Out

Statement_Assign ::= LHS Expression

REQUIRE: LHS.Type == Expression.Type
StatementAssign.isCartesian <= LHS.isCartesian

Statement_In ::= name Source

Generate code to get value from the source and store it in variable name.
For Assignment 5, the only source that needs to be handled is the command line.

Visit source to leave string representation of the value on top of stack
Convert to a value of correct type: If name.type == INTEGER generate code to invoke
Java.lang.Integer.parseInt. If BOOLEAN, invoke java/lang/Boolean.parseBoolean

TODO: Handling IMAGE type left for assignment 6.

Statement_In.Declaration <= name.Declaration

~~REQUIRE: if (name.Declaration != null) & (name.type == Source.type)~~

Statement_Out ::= name Sink

For INTEGERS and BOOLEANS, the only "sink" is the screen, so generate code to print to the console here. Use java.io.PrintStream .println. This is a virtual method, you can use the static field PrintStream "out" from class java.lang.System as the object.

TODO: handle images in Assignment 6

Expression ::= Expression_Binary | Expression_BooleanLit | Expression_Conditional | Expression_FunctionApp | Expression_FunctionAppWithExprArg | Expression_FunctionAppWithIndexArg | Expression_Ident | Expression_IntLit | Expression_PixelSelector | Expression_PredefinedName _ Expression_Unary

For each expression kind, generate code to leave the value of the expression on top of the stack.

Expression_Binary ::= Expression_0 op Expression_1

Generate code to evaluate the expression and leave the value on top of the stack.

Visiting the nodes for Expression_0 and Expression_1 will generate code to leave those values on the stack. Then just generate code to perform the op.

Expression_BooleanLit ::= value

Generate code to leave the value of the literal on top of the stack

Expression_Conditional ::= $\text{Expression}_{\text{condition}}$ $\text{Expression}_{\text{true}}$ $\text{Expression}_{\text{false}}$

Generate code to evaluate the $\text{Expression}_{\text{condition}}$ and depending on its Value, to leave the value of either $\text{Expression}_{\text{true}}$ or $\text{Expression}_{\text{false}}$ on top of the stack.
Hint: you will need to use labels, a conditional instruction, and goto.

Expression_FunctionApp ::= Expression_FunctionAppWithExprArg

| Expression_FunctionAppWithIndexArg

TODO in assignment 6

Expression_FunctionAppWithExprArg ::= function Expression

TODO in assignment 6

Expression_FunctionAppWithIndexArg ::= function Index

TODO in assignment 6

Expression_Ident ::= name

Generate code to get the value of the variable and leave it on top of the stack.

Expression_IntLit ::= value

Generate code to leave constant on stack.

Expression_PixelSelector ::= name Index

TODO Assignment 6

Expression_PredefinedName ::= predefNameKind

~~Generate code to get the value of the variable and leave it on top of the stack.~~

TODO Assignment 6 ⇐ NEW

Expression_Unary ::= op Expression

Generate code to evaluate the unary expression and leave its value on top of the stack. Which code is generated will depend on the operator. If the op is OP_PLUS, the value that should be left on the stack is just the value of Expression

Index ::= Expression_0 Expression_1

TODO Assignment 6

LHS ::= name Index

If LHS.Type is INTEGER or BOOLEAN, generate code to

store the value on top of the stack in variable name.

TODO Assignment 6: handle case where LHS.Type is IMAGE

Sink ::= Sink_Ident | Sink_SCREEN

Sink_Ident ::= name

TODO Assignment 6

Sink_SCREEN ::= SCREEN

TODO Assignment 6

Source ::= Source_CommandLineParam | Source_Ident | Source_StringLiteral

Source_CommandLineParam ::= $\text{Expression}_{\text{paramNum}}$

Generate code to evaluate the expression and use aaload to read the element from the command line array using the expression value as the index. The command line array is the String[] args param passed to main.

Source_Ident ::= name

TODO in Assignment 6

Source_StringLiteral ::= fileOrURL

TODO in Assignment 6

Corrections to previous assignment.

Change the type checking of

Statement_In ::= name Source

Statement_In.Declaration <= name.Declaration
~~REQUIRE: if (name.Declaration != null) & (name.type == Source.type)~~

Requirements:

For grading purpose, it is essential that you insert calls to CodeGenUtils.genLog(..) and CodeGenUtils.genLogTOS(...) at locations indicated in the comments of the provided class CodeGenVisitor.

The provided class CodeGenVisitorTest contains several test cases that should be useful to you. The empty program test should pass immediately. The others should pass in a completed assignment. These are not complete, you will need additional test cases.

Turn in a jar file containing your source code for CodeGenVisitor.java, CodeGenVisitorTest.java, TypeCheckVisitor.java, Parser.java, Scanner.java, all of the AST classes, TypeUtils.java, RuntimeLog.java, CodeGenUtils.java and any classes that you have added.

Your CodeGenTest will not be graded, but may be looked at in case of academic honesty issues. We will subject your parser 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_hw5.jar***

Comments and Suggestions

- IMPORTANT: Review the lectures on the JVM and the assignment before you begin.
- 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, Parser, and TypeChecker after they have been graded.

- Hint: be careful to keep the throw UnsupportedOperationException in place until a feature is implemented. This will save you a tremendous amount of time if you accidentally invoke a test program containing an unimplemented feature.