**COP 5556 // Project #6 // Fall 2018**

|  |  |
| --- | --- |
| **Date Assigned:** | November 6, 2018 |
| **Date Due:** | November 20, 2018 |

# Submission Format

You will submit a soft copy of your solution using e-Learning ( <http://elearning.ufl.edu> ) by the end of the day ( 23:59 / 11:59 PM ) on the assigned date ( **November 20** ). Save your solution as a **jar** file and name the file **p6** ( p6.jar ).

# Assignment

At the top of every solution file you submit this semester include: your name, the assignment number, and the date due. PLPCodeGen.java, PLPCodeGenTest.java, PLPRuntimeLog.java, PLPCodeGenUtils.java has been provided. You will need to complete the implementation of PLPCodeGen.java. Any additional JUnit testing you would like to perform can be included in PLPCodeGenTest.java.

You will need to add a getter method like below to Expression.java:

public Type getType() {

return type;

}

PLPCodeGen.java will generate code for the test cases given in PLPCodeGenTest. Some of the code has been implemented for you as an example. You will need to extend this functionality, by modifying the existing code base. You will not modify the class structure or existing method signatures. If you find additional classes or methods helpful, you are allowed to implement them. Note, one example of what is given, is the implementation for printing an int value.

There are methods in PLPCodeGenUtils that generate code to print or log the top of the stack (without consuming it) and print or log a string. These may be useful for debugging. If you use them that way, they should be turned on with DEVEL. You can see examples that use GRADE in visitProgram and visitPrintStatement. In our test script, GRADE will be true, and DEVEL will be false. If you use these methods with GRADE in a way other than exactly as specified, the test cases will fail.

To ensure that we will be able to compile and run your submissions, test your code on one of the UF CISE servers, like you did for previous projects. We suggest trying to work incrementally and reviewing the lecture on JVM before you begin.

Hint: If you are getting errors in visitMaxs, you may want to try temporarily replacing the parameter in the ClassWriter constructor to 0. The result will not be a well-formed classfile, but you can at least see which instructions have been generated.

Submit a jar file containing your source code for:

* PLPCodeGen.java
* PLPCodeGenTest.java
* PLPRuntimeLog.java
* PLPCodeGenUtils.java
* PLPTypeChecker.java
* PLPTypeCheckerTest.java
* PLPParser.java
* PLPScanner.java
* all of the AST classes
* PLPTypes.java
* any classes you may have added

|  |  |
| --- | --- |
| **Abstract Syntax** | **Code generation** |
| Program IDENTIFIER Block | Generate code for a class whose name is given by the Program.name attribute  Create the public static void main(String[] args) method and add local variable args  Block becomes the body of this class’s public static void main(String[] args) method |
| Block ( Declaration | Statement )\* |  |
|  |  |
| Declaration VariableDeclaration | VariableListDeclaration |  |
| VariableDeclaration Type IDENTIFIER ( | Expression ) | Add a local variable with the Declaration’s name and type to the class. Add an attribute to the Declaration for the assigned local variable slot number.  If Expression is not null, visit the Expression to generate code to evaluate them and leave their vaule on the stack. |
| VariableListDeclaration Type IDENTIFIER IDENTIFIER + | For each identifier, add a local variable with the declaration’s name and type to the class. Add an attribute to the Declaration for the assigned local variable slot number. |
|  |  |
| Type int | float | boolean | char | string | |  |  | | --- | --- | | Type | Java  JVM type | | int | int  I | | float | float  F | | boolean | boolean  Z | | char | char  C | | String | java.lang.String  LJava/lang/String; | |
|  |  |
|  |  |
| Statement IfStatement | AssignmentStatement | SleepStatement | PrintStatement | WhileStatement |  |
|  |  |
|  |  |
| AssignmentStatement LHS Expression | Visit the Expression to generate code to leave the expressions value on top of the stack.  Visit LHS to generate code to store the top of the stack into the indicated variable. |
| WhileStatement Expression Block | Generate code to implement a while loop |
| IfStatement Expression Block | Generate code to implement an if statement |
| PrintStatement Expression | Visit the Expression to generate code to evaluate the expression and leave it value on top of the stack.  Output the value to the console by invoking the println method of java/io/PrintStream. Note that this method is overloaded, you will need to call it with the correct type.  IMPORTANT: For all types, the CodeGenUtils.genLogTOS(GRADE, mv, type); method should be called before the top of the stack is consumed. This log will be used for grading, so it is crucial that this be done. |
| SleepStatement Expression | The value of the expression is the number of milliseconds that the program should sleep. Use java.lang.Thread.sleep |
| LHS IDENTIFIER | Generate code to store the value already on top of the stack in the corresponding variable. |
| Expression ExpressionBinary | ExpressionConditional | FunctionWithArg | ExpressionUnary | ExpressionIdent | ExpressionIntegerLiteral | ExpressionBooleanLiteral | ExpressionFloatLiteral | ExpressionCharLiteral | ExpressionStringLiteral | Always handle Expressions by generating code to evaluate the expression and leave its value on top of the stack.  Other than a few hints and clarifications, how to do this is left for you to figure out.  It may be useful for you to write little programs in Java with equivalent semantics and look at the code generated by the Java compiler. You may also use the asmifier eclipse plugin, or command line tool in class org.objectweb.asm.util.asmifier to print the ASM code that would generate a given class. |
| ExpressionConditional Expression0 Expression1 Expression2 |  |
| ExpressionBinary Expression0 op Expression1 | Implement all binary operators in our language.  Hint: Implement POWER by converting both arguments to double, invoking java/lang/Math.pow, and converting the result back to the appropriate type. The others should be straightforward. |
| ExpressionUnary Op Expression | Implement all of the unary operators in our language.  Hint: Operator ! applied to an integer, should flip all the bits including the sign bit. |
| ExpressionIdent |  |
| ExpressionIntegerLiteral |  |
| ExpressionBooleanLiteral |  |
| ExpressionFloatLiteral |  |
| ExpressionCharLiteral |  |
| ExpressionStringLiteral |  |
|  |  |
| FunctionWithArg  FunctionName Expression | For sin, cos, atan, log, abs: use functions in java.lang.Math.  Some of these functions expect a double argument and return a double value, so you will need to cast to and from float.  You may find it easier to write a wrapper routine in Java to do this and invoke your function instead of invoking the java.lang.Math function directly.  function int converts a float to an int (JVM instruction F2I) or does nothing if the type is already int.  function float converts and int to float (JVM instruction I2F) or does nothing if the type is already float. |
| FunctionName sin | cos | atan | abs | log | int | float |  |