Introduction to Compiler Construction

Assignment 5 (Type Checking and Code Generation) Discussion

Problem 1 (Operators)

Add support for the following operators:

`-=` `*=` `/=` `%=` `!=` `>=` `<` `||` `++` `--`

Modify `analyze()` and `codegen()` in `JNegateOp` and `JUnaryPlusOp`; the operand can be an int, long, or double

Modify `analyze()` and `codegen()` in `JPlusOp`, `JSubtractOp`, `JMultiplyOp`, `JDivideOp`, and `JRemainderOp`; the operands can be an ints, longs, or doubles

Modify `analyze()` and `codegen()` in `JPlusAssignOp`; the operands can be an ints, longs, or doubles

Modify `analyze()` in `JComparisonExpression`; the operands can be an ints, longs, or doubles

Problem 1 (Operators)

Implement `analyze()` and `codegen()` in `JMinusAssignOp`, `JStarAssignOp`, `JDivAssignOp`, and `JRemAssignOp`; the operands can be ints, longs, or doubles

Implement `analyze()` and `codegen()` in `JNotEqualOp`

Implement `codegen()` in `JGreaterEqualOp` and `JLessThanOp`; the operands can be an ints, longs, or doubles

Implement `analyze()` and `codegen()` in `JLogicalOrOp`

Implement `analyze()` and `codegen()` in `JPostIncrementOp` and `JPreDecrementOp`; the operand must be an int

Problem 1 (Operators)

Testing

>_ ~/workspace/j--

```
$ ant
$ ./bin/j-- codegen/Operators.java
$ java Operators 23 3
a               : 23
b               : 3
a -= b          : 20
a *= b          : 60
a /= b          : 20
a %= b          : 2
a != b          : true
a >= b          : false
a < b           : true
a < 0 || b < 0 : false
--a             : 1
b++             : 3
```

Problem 2 (Long and Double Basic Types)

Add support for the `long` and `double` basic types

Implement `analyze()` and `codegen()` in `JLiteralLong` and `JLiteralDouble`

Modify `partialCodegen()` in `JMethodDeclaration`

Modify `analyze()` in `JConstructorDeclaration`, `JMethodDeclaration`, and `JVariableDeclaration` to skip an offset for longs and doubles

Modify `codegen()` in `JReturnStatement`

Modify the 1-argument `codegen()` method and the `codegenStore()` method in `JVariable`

Modify 1-argument `codegen()`, `codegenLoadLhsRvalue`, and `codegenStore()` in `JArrayExpression`

Modify `codegen()` in `JArrayInitializer`

Problem 2 (Long and Double Basic Types)

Testing

```
>_ ~/workspace/j--
```

```
$ ant
$ ./bin/j-- codegen/Factorial.java
$ java Factorial 7
5040
$ ./bin/j-- codegen/Quadratic.java
$ java Quadratic 1 -5 6
3.0 2.0
```

Problem 3 (For Statement)

Add support for a for statement

Create a new `LocalContext` with `context` as the parent

Analyze the init in the new context

Analyze the condition in the new context and make sure it's a boolean

Analyze the update in the new context

Analyze the body in the new context

Implement `codegen()`

Problem 3 (For Statement)

Testing

```
>_ ~/workspace/j--
```

```
$ ant
$ ./bin/j-- codegen/ForStatement.java
$ java ForStatement 100
5050
$ ./bin/j-- codege/Stats.java
$ java Stats
Mean   = 5.5
Stddev = 2.8722813232690143
```

Problem 4 (Break Statement)

Add support for a break statement

Create an empty stack in `JMember` to keep track of the surrounding control-flow statement

```
public static Stack<JStatement> enclosingStatement = new Stack<JStatement>();
```

Declare two instance variables in each control-flow statement (do, while, for, and switch): `boolean hasBreak` and `String breakLabel`

Each control-flow statement (do, while, for, and switch), during analysis, must push `this` onto `JMember.enclosingStatement` upon entry, and pop it upon exit

Each control-flow statement (do, while, for, and switch), during codegen, must set `breakLabel` to an appropriate label if `hasBreak` is `true`, and add the label at the appropriate place

Declare an instance variable `JStatement enclosingStatement` in `JBreakStatement`, and during analysis, set it to the value at the top of `JMember.enclosingStatement` (use `peek()`); then set the enclosing statement's `hasBreak` variable to `true`

During codegen in `JBreakStatement`, access the break label via the enclosing statement, and generate an unconditional jump to that label

Problem 4 (Break Statement)

Testing

>_ ~/workspace/j--

```
$ ant
$ ./bin/j-- codegen/BreakStatement.java
$ java BreakStatement 1000
168
```

Problem 5 (Continue Statement)

Add support for a continue statement

Declare two instance variables in each control-flow statement (do, while, and for): `boolean hasContinue` and `String continueLabel`

Each control-flow statement (do, while, and for), during codegen, must set `continueLabel` to an appropriate label if `hasContinue` is `true`, and add the label at the appropriate place

During analysis in `JContinueStatement`, set the enclosing statement's `hasContinue` variable to `true`

During codegen in `JContinueStatement`, access the continue label via the enclosing statement, and generate an unconditional jump to that label

Problem 5 (Continue Statement)

Testing

>_ ~/workspace/j--

```
$ ant
$ ./bin/j-- codegen/ContinueStatement.java
$ java ContinueStatement 100
3.121594652591011
```

Problem 6 (Switch Statement)

Add support for a switch statement

Code to decide which instruction (`TABLESWITCH` or `LOOKUPSWITCH`) to emit:

```
long tableSpaceCost = 5 + hi - lo;
long tableTimeCost = 3;
long lookupSpaceCost = 3 + 2 * nLabels;
long lookupTimeCost = nLabels;
int opcode = nLabels > 0 && (tableSpaceCost + 3 * tableTimeCost <= lookupSpaceCost + 3 * lookupTimeCost) ?
                TABLESWITCH : LOOKUPSWITCH;
```

Where `hi` is the highest case label value, `lo` is the lowest case label value, and `nlabels` are the total real case labels in the switch statement.

Analyze the condition and make sure it is an integer

Anayze the case expressions and make sure they are integer literals

Create a new `LocalContext` with `context` as the parent, and analyze the statements in each case group in the new context

Problem 6 (Switch Statement)

In `codegen()` decide which instruction (`TABLESWITCH` or `LOOKUPSWITCH`) to emit using the above heuristic

Call the appropriate `CLEmitter` method to emit that instruction — you will first need to gather all the information that must be passed as arguments to the method

Generate code for the case group statements, adding labels at the appropriate places

Consult `$j/j--/tests/GenTableSwitch.java` and `$j/j--/tests/GenLookupSwitch.java` for more hints on codegen

Problem 6 (Switch Statement)

Testing

```
>_ ~/workspace/j--
```

```
$ ant
$ ./bin/j-- codegen/SwitchStatement.java
$ java SwitchStatement
Queen of Hearts
$ java SwitchStatement
Jack of Spades
```