COP5555 Spring 2012
Project 4

Assigned Feb 2
Due: March 20 at 3pm

Follow the instructions carefully. Test cases that fail due to failure to follow instructions will not be given credit and no regrades will be allowed.

1. Implement a SymbolTable class that can correctly handle nested scopes. The LeBlanc-Cook symbol table described in class is appropriate. Fill in the attached file. This class will be tested separately.
2. Implement a visitor class called **ContextCheckVisitor** in package `edu.ufl.cise.cop5555.sp12.context` to traverse your AST from Project 3 and check context conditions. ContextCheckVisitor implements the ASTVisitor interface, thus must provide implementations of all methods in the interface.

   The rules are given in the attachment In pseudo-attribute grammar style. Check each condition in visit method of the corresponding AST node. If a condition is violated, throw a new ContextException and pass it the ast where the error occurred, and a String containing an error message. The contents of the error message will not be graded, but for your own benefit, you should make them descriptive. The correct ast node will be checked. The ContextException class is attached.

Your visitor class should complete the following. (Your IDE should create the stub methods you need to implement for you.)

```
package edu.ufl.cise.cop5555.sp12.context;

import ….

public class ContextCheckVisitor implements ASTVisitor {

 //TODO  implement methods of interface

}
```

Your visitor class can be used as in the following code snippet

```
//set up scanner and parser as in previous projects
AST ast = parser.parse();  //get AST from project 3
ASTVisitor contextChecker = new ContextCheckVisitor();
ast.visit(contextChecker,null);
```

In your implementation you may add fields and methods to the AST classes and you may also change the visibility of the existing fields and methods. You should not remove anything or change signatures of existing methods.

Here is an example

```
@Override
public Object visitAssignExprCommand(AssignExprCommand assignExprCommand,
            Object arg) throws Exception {
      Type lhsType = (Type) assignExprCommand.lValue.visit(this,arg);
      Type exprType = (Type) assignExprCommand.expression.visit(this,arg);
      check(lhsType.equals(exprType), assignExprCommand,
                "incompatible types in assignment");
      return null;
}
```

To check a an AssignExprCommand, we need to check that the types of the left and right hand sides are the same. This is done by visiting both the left hand side (the lValue) and the right hand side (the expression), and arranging for their visit methods return their type. Then check that the types are equal. For convenience, I have a check method that throws a ContextException containing the assignExprCommand object and the given message if the first argument is not true. The visitAssignExprCommand itself does not return a value, it just checks that it (and its children) are correct.

This will not be checked it is not obviously needed in project 4, but it is recommended for convenience later that you add a Type field to the Expression class and set it when you visit each expression type.

Turn in a jar file called P4.jar containing all of the sources needed to scan, parse, and perform context checking on the code, including those from previous projects. We will recompile and call the public methods from our test program. Do not change the signatures of these methods or the package declarations or you will break the test script.