/\*\*

\* Private constructor so this utility class cannot be instantiated.

\*/

private CountPrimitiveCalls() {

}

/\*\*

\* Reports the number of calls to primitive instructions (move, turnleft,

\* turnright, infect, skip) in a given {@code Statement}.

\*

\* @param s

\* the {@code Statement}

\* @return the number of calls to primitive instructions in {@code s}

\* @ensures

\*

\* <pre>

\* countOfPrimitiveCalls =

\* [number of calls to primitive instructions in s]

\* </pre>

\*/

public static int countOfPrimitiveCalls(Statement s) {

int count = 0;

switch (s.kind()) {

case BLOCK: {

/\*

\* Add up the number of calls to primitive instructions in each

\* nested statement in the BLOCK.

\*/

int length = s.lengthOfBlock();

for (int i = 0; i < length; i++) {

Statement subStatement = s.removeFromBlock(i);

count += countOfPrimitiveCalls(subStatement);

s.addToBlock(i, subStatement);

}

break;

}

case IF: {

/\*

\* Find the number of calls to primitive instructions in the

\* body of the IF.

\*/

Statement subStatement = s.newInstance();

Statement.Condition condition = s.disassembleIf(subStatement);

count = countOfPrimitiveCalls(subStatement);

s.assembleIf(condition, subStatement);

break;

}

case IF\_ELSE: {

/\*

\* Add up the number of calls to primitive instructions in the

\* "then" and "else" bodies of the IF\_ELSE.

\*/

Statement subStatementIf = s.newInstance();

Statement subStatementElse = s.newInstance();

Statement.Condition condition = s.disassembleIfElse(subStatementIf, subStatementElse);

count = countOfPrimitiveCalls(subStatementIf) + countOfPrimitiveCalls(subStatementElse);

s.assembleIfElse(condition, subStatementIf, subStatementElse);

break;

}

case WHILE: {

/\*

\* Find the number of calls to primitive instructions in the

\* body of the WHILE.

\*/

Statement subStatement = s.newInstance();

Statement.Condition condition = s.disassembleWhile(subStatement);

count = countOfPrimitiveCalls(subStatement);

s.assembleWhile(condition, subStatement);

break;

}

case CALL: {

/\*

\* This is a leaf: the count can only be 1 or 0. Determine

\* whether this is a call to a primitive instruction or not.

\*/

String callLabel = s.disassembleCall();

if (callLabel.equals("turnright") || callLabel.equals("move")

|| callLabel.equals("infect") || callLabel.equals("turnleft")

|| callLabel.equals("skip")) {

count++;

}

s.assembleCall(callLabel);

break;

}

default: {

// this will never happen...can you explain why?

break;

}

}

return count;

}