```java
/**
 * Refactors the given {@code Statement} by renaming every occurrence of
 * instruction {@code oldName} to {@code newName}. Every other statement is
 * left unmodified.
 *
 * @param s
 *            the {@code Statement}
 * @param oldName
 *            the name of the instruction to be renamed
 * @param newName
 *            the new name of the renamed instruction
 * @updates s
 * @requires [newName is a valid IDENTIFIER]
 * @ensures <pre>
 * s = [#s refactored so that every occurrence of instruction oldName
 *   is replaced by newName]
 * </pre>
 */
public static void renameInstruction(Statement s, String oldName,
        String newName) {
            // Will ask about this in office hours
        }
/**
 * Refactors the given {@code Program} by renaming instruction
 * {@code oldName}, and every call to it, to {@code newName}. Everything
 * else is left unmodified.
 *
 * @param p
 *            the {@code Program}
 * @param oldName
 *            the name of the instruction to be renamed
 * @param newName
 *            the new name of the renamed instruction
 * @updates p
 * @requires <pre>
 * oldName is in DOMAIN(p.context)  and
 * [newName is a valid IDENTIFIER]  and
 * newName is not in DOMAIN(p.context)
 * </pre>
 * @ensures <pre>
 * p = [#p refactored so that instruction oldName and every call
 *   to it are replaced by newName]
 * </pre>
 */
public static void renameInstruction(Program p, String oldName,
        String newName) {
            // TODO - fill in body
        Map<String, Statement> contex = p.newContext();
        p.swapContext(contex);
        while (  < contex.size()) {
            Map.Pair<String, Statement> temp = contex.removeAny();
            if(temp.key().equals(oldName)){
            p.add(newName, temp.value());
            }else{
            p.add(temp.key(), temp.value());
            }
        }
    }


Program p = new Program1();
Map<String, Statement> context = p.newContext();
Statement block = p.newBody();
Statement s = block.newInstance();
p.setName("Get-to-Edge-and-Wait-for-Infection");
s.assembleCall("walk");
block.addToBlock( , s);
s.assembleCall("run");
block.addToBlock(block.lengthOfBlock(), s);
```

```
s.assembleWhile(Condition.NEXT_IS_NOT_WALL, block);
block.addToBlock( , s);
p.swapBody(block);
s.assembleCall("move");
block.addToBlock( , s);
s.assembleCall("move");
block.addToBlock(block.lengthOfBlock(), s);
context.add("run", block);
s.assembleCall("move");
block = block.newInstance();
block.addToBlock( , s);
context.add("walk", block);
p.swapContext(context);
// I did not know how to fully do this
```