```
 1 package test;
 2
 3 import java.io.ByteArrayOutputStream;
 4 import java.io.PrintWriter;
 5 import java.io.StringReader;
 6 import java.lang.reflect.Method;
 7 import java.net.URL;
 8 import java.net.URLClassLoader;
 9 import java.util.HashMap;
10 import java.util.Map;
11
12 import org.junit.Test;
13
14 import backend.ProgramCodeGenerator;
15
16 import org.junit.Assert;
17 import lexer.Lexer;
18 import parser.Parser;
19 import soot.Printer;
20 import soot.SootClass;
21 import soot.jimple.JasminClass;
22 import soot.util.JasminOutputStream;
23 import ast.List;
24 import ast.Module;
25 import ast.Program;
26
27 /**
28  * System tests for the compiler: compiles a given program to Java bytecode, then
   immediately
29  * loads it into the running JVM and executes it.
30  */
31 public class CompilerTests {
32     // set this flag to true to dump generated Jimple code to standard output
33     private static final boolean DEBUG = false;
34
35     /**
36      * A simple class loader that allows us to directly load compiled classes.
37      */
38     private static class CompiledClassLoader extends URLClassLoader {
39         private final Map<String, byte[]> classes = new HashMap<String, byte[]>();
40
41         public CompiledClassLoader() {
42             super(new URL[0], CompilerTests.class.getClassLoader());
43         }
44
45         public void addClass(String name, byte[] code) {
46             classes.put(name, code);
47         }
48
49         @Override
50         protected Class<?> findClass(String name) throws ClassNotFoundException {
51             if(classes.containsKey(name)) {
52                 byte[] code = classes.get(name);
53                 return defineClass(name, code, 0, code.length);
54             }
```

```
55              return super.findClass(name);
56          }
57      }
58
59      /**
60       * Test runner class.
61       *
62       * @param modules_src Array of strings, representing the source code of the
   program modules
63       * @param main_module the name of the main module
64       * @param main_function the name of the main function
65       * @param parm_types the parameter types of the main function
66       * @param args arguments to pass to the main method
67       * @param expected expected result
68       */
69      private void runtest(String[] modules_src, String main_module, String
   main_function, Class<?>[] parm_types, Object[] args, Object expected) {
70          try {
71              List<Module> modules = new List<Module>();
72              for(String module_src : modules_src) {
73                  Parser parser = new Parser();
74                  Module module = (Module)parser.parse(new Lexer(new StringReader(
   module_src)));
75                  modules.add(module);
76              }
77              Program prog = new Program(modules);
78
79              prog.namecheck();
80              prog.typecheck();
81              prog.flowcheck();
82              if(prog.hasErrors()) {
83                  Assert.fail(prog.getErrors().iterator().next().toString());
84              }
85
86              CompiledClassLoader loader = new CompiledClassLoader();
87              try {
88                  for(SootClass klass : new ProgramCodeGenerator().generate(prog))
   {
89                      if(DEBUG) {
90                          PrintWriter stdout_pw = new PrintWriter(System.out);
91                          Printer.v().printTo(klass, stdout_pw);
92                          stdout_pw.flush();
93                      }
94
95                      String name = klass.getName();
96                      ByteArrayOutputStream baos = new ByteArrayOutputStream();
97                      PrintWriter pw = new PrintWriter(new JasminOutputStream(baos)
   );
98                      new JasminClass(klass).print(pw);
99                      pw.flush();
100                     loader.addClass(name, baos.toByteArray());
101                 }
102
103                 Class<?> testclass = loader.loadClass(main_module);
104                 Method method = testclass.getMethod(main_function, parm_types);
```

```
105                 Object actual = method.invoke(null, args);
106                 if(!method.getReturnType().equals(void.class))
107                     Assert.assertEquals(expected, actual);
108             } finally {
109                 loader.close();
110             }
111         } catch(Exception e) {
112             e.printStackTrace();
113             Assert.fail(e.getMessage());
114         } catch(ClassFormatError e) {
115             e.printStackTrace();
116             Assert.fail(e.getMessage());
117         }
118     }
119 
120     /** Convenience wrapper for runtest with only a single module. Other
    arguments are the same .*/
121     private void runtest(String string, String classname, String methodname,
    Class<?>[] parmTypes, Object[] args, Object expected) {
122         runtest(new String[] { string }, classname, methodname, parmTypes, args,
    expected);
123     }
124 
125     @Test public void testAddition() {
126         runtest("module Test {" +
127                 "  public int f() {" +
128                 "    return 23+19;" +
129                 "  }" +
130                 "}",
131                 "Test",
132                 "f",
133                 new Class<?>[0],
134                 new Object[0],
135                 42);
136     }
137 
138     @Test public void testSubtraction() {
139         runtest("module Test {" +
140                 "  public int f() {" +
141                 "    return 42-19;" +
142                 "  }" +
143                 "}",
144                 "Test",
145                 "f",
146                 new Class<?>[0],
147                 new Object[0],
148                 23);
149     }
150 
151     @Test public void testMultiplication() {
152         runtest("module Test {" +
153                 "  public int f() {" +
154                 "    return 2*21;" +
155                 "  }" +
156                 "}",
```

```
157                 "Test",
158                 "f",
159                 new Class<?>[0],
160                 new Object[0],
161                 42);
162     }
163 
164     @Test public void testDivision() {
165         runtest("module Test {" +
166                 "  public int f() {" +
167                 "    return 42/2;" +
168                 "  }" +
169                 "}",
170                 "Test",
171                 "f",
172                 new Class<?>[0],
173                 new Object[0],
174                 21);
175     }
176 
177     @Test public void testPrecedence() {
178         runtest("module Test {" +
179                 "  public int f() {" +
180                 "    return 2+42/2;" +
181                 "  }" +
182                 "}",
183                 "Test",
184                 "f",
185                 new Class<?>[0],
186                 new Object[0],
187                 23);
188     }
189 
190      @Test public void testParentheses() {
191         runtest("module Test {" +
192                 "  public int f() {" +
193                 "    return (2+42)/2;" +
194                 "  }" +
195                 "}",
196                 "Test",
197                 "f",
198                 new Class<?>[0],
199                 new Object[0],
200                 22);
201     }
202 
203     @Test public void testNegation() {
204         runtest("module Test {" +
205                 "  public int f() {" +
206                 "    return -42;" +
207                 "  }" +
208                 "}",
209                 "Test",
210                 "f",
211                 new Class<?>[0],
```

```
212                 new Object[0],
213                 -42);
214     }
215
216     @Test public void testString() {
217         runtest("module Test {" +
218                 "  public String f() {" +
219                 "  String foo;" +
220                 "  foo = \"qwerty\";" +
221                 "    return foo;" +
222                 "  }" +
223                 "}",
224                 "Test",
225                 "f",
226                 new Class<?>[0],
227                 new Object[0],
228                 "qwerty");
229     }
230
231     @Test public void testint() {
232         runtest("module Test {" +
233                 "  public int f() {" +
234                 "  int foo ;" +
235                 "  foo = 123;" +
236                 "    return foo;" +
237                 "  }" +
238                 "}",
239                 "Test",
240                 "f",
241                 new Class<?>[0],
242                 new Object[0],
243                 123);
244     }
245
246     @Test public void testintAddition() {
247         runtest("module Test {" +
248                 "  public int f() {" +
249                 "  int foo ;" +
250                 "  foo = 123+2;" +
251                 "    return foo + 2;" +
252                 "  }" +
253                 "}",
254                 "Test",
255                 "f",
256                 new Class<?>[0],
257                 new Object[0],
258                 127);
259     }
260
261     @Test public void testBoolean() {
262         runtest("module Test {" +
263                 "  public boolean f() {" +
264                 "  boolean foo ;" +
265                 "  foo = true;" +
266                 "    return foo;" +
```

```
267                 "  }" +
268                 "}",
269                 "Test",
270                 "f",
271                 new Class<?>[0],
272                 new Object[0],
273                 true);
274     }
275 
276     @Test public void testBinaryExper() {
277         runtest("module Test {" +
278                 " public int a(){" +
279                 "   return 5;" +
280                 " }" +
281                 "  public boolean f() {" +
282                 "  int foo ;" +
283                 "  foo = 123;" +
284                 "    return foo > a();" +
285                 "  }" +
286                 "}",
287                 "Test",
288                 "f",
289                 new Class<?>[0],
290                 new Object[0],
291                 true);
292     }
293 
294     @Test public void testNeqExpr() {
295         runtest("module Test {" +
296                 " public int a(){" +
297                 "   return 5;" +
298                 " }" +
299                 "  public boolean f() {" +
300                 "  int foo ;" +
301                 "  foo = 123;" +
302                 "    return foo != a();" +
303                 "  }" +
304                 "}",
305                 "Test",
306                 "f",
307                 new Class<?>[0],
308                 new Object[0],
309                 true);
310     }
311 
312     @Test public void testArrayIndx() {
313         runtest("module Test {" +
314                 "  public int f() {" +
315                 "  int[] intArray;" +
316                 "  intArray = int[3];" +
317                 "  intArray[0] = 123;" +
318                 "    return intArray[0];" +
319                 "  }" +
320                 "}",
321                 "Test",
```

```
322                 "f",
323                 new Class<?>[0],
324                 new Object[0],
325                 123);
326     }
327 
328     @Test public void testArrayFunc() {
329         runtest("module Test {" +
330                 " public int a(){" +
331                 "   return 0;" +
332                 " }" +
333                 "  public int f() {" +
334                 "  int[] intArray;" +
335                 "  intArray = int[3];" +
336                 "  intArray[0] = 123;" +
337                 "    return intArray[a()];" +
338                 "  }" +
339                 "}",
340                 "Test",
341                 "f",
342                 new Class<?>[0],
343                 new Object[0],
344                 123);
345     }
346 
347     @Test public void testMod() {
348         runtest("module Test {" +
349                 "  public int f() {" +
350                 "  int foo;" +
351                 "  foo = 5 % 2;" +
352                 "    return foo;" +
353                 "  }" +
354                 "}",
355                 "Test",
356                 "f",
357                 new Class<?>[0],
358                 new Object[0],
359                 1);
360     }
361 
362     @Test public void testCallParam() {
363         runtest("module Test {" +
364                 "  public int f(int x) {" +
365                 "  if (x <= 5){" +
366                 "   return x;" +
367                 "   }else{" +
368                 "       return f(x-1); " +
369                 "   }" +
370                 "  }" +
371                 "  public int foo() {" +
372                 "  return f(10);" +
373                 "  }" +
374                 "}",
375                 "Test",
376                 "foo",
```

```
377                 new Class<?>[0],
378                 new Object[0],
379                 5);
380     }
381 
382     @Test public void testWhile() {
383         runtest("module Test {" +
384                 "  public int f() {" +
385                 "int x;"
386                 + "x = 0;"
387                 + "while(x<100){"
388                 + " x = x + 1;"
389                 + " if (x > 10) {"
390                 + "  "
391                 + "} else {"
392                 + "   x = x + 1;"
393                 + "}"
394                 + "}"
395                 + "return x;"
396                 + "}"
397                 + "}",
398                 "Test",
399                 "f",
400                 new Class<?>[0],
401                 new Object[0],
402                 100);
403     }
404 
405     @Test public void testWhileBreak() {
406         runtest("module Test {" +
407                 "  public int f() {" +
408                 "int x;"
409                 + "x = 0;"
410                 + "while(x<100){"
411                 + " x = x + 1;"
412                 + " if (x > 10) {"
413                 + "  break;"
414                 + "}"
415                 + "}"
416                 + "return x;"
417                 + "}"
418                 + "}",
419                 "Test",
420                 "f",
421                 new Class<?>[0],
422                 new Object[0],
423                 11);
424     }
425 }
426 
```