Exercise 8: Indirect Recursion

MicroJava supports direct recursion, where a method can call itself. However, it does not support indirect recursion, where, for example, a method foo() calls a method bar() and that method calls foo() again. Why is indirect recursion not supported by MicroJava?

Solution

The MicroJava compiler is a single-pass compiler that processes a program strictly sequentially. If it encounters the use of a name, this name must have been declared before. However, this is not possible with indirect recursion.

```
void foo() {
   if (...) bar();
   ...
}

void bar() {
   if (...) foo();
   ...
}
```

When processing foo(), bar() is not yet declared. If the two methods were declared in reverse order, you would have the same problem: foo() would not yet be declared when bar() is processed.

Multi-pass compilers, on the other hand, build up an internal program representation (e.g. an abstract syntax tree). In the first pass, the declarations are processed and entered into the symbol table, but the names in the statements are not yet looked up in the symbol table. The statements are then processed in a second pass. At this point, all declarations are already known, so that indirect recursion is possible.

Single-pass compilers often solve the problem of indirect recursion via "forward declarations". The signature of a method can be declared without code, for example:

```
void bar(); forward;

void foo() {
   if (...) bar();
   ...
}

void bar() {
   if (...) foo();
   ...
}
```

In this case, the declaration of bar() is already known when the statements in foo() are processed. However, when bar() is then actually declared, you must check whether its parameters match the forward declaration. Forward declarations are not supported by MicroJava.