Using the Scala library `pilib`, we can implement process calculus terms as Scala programs, using a syntax fairly close to the original. An archive containing the library and some stubs for the following exercises can be found on the course web page.

Because of problems with the `include` directive, the entire library must be recompiled together with your programs. This is done automatically if you use the makefile on the web site. To compile the program `testProgram.scala` you write this command standing in the directory where the makefile (and `testProgram.scala`) is.

```
make FILE=testProgram.scala
```

Assuming that it is compiled, you can run the module `testModule` by executing

```
make play ENTRY_MODULE=testModule
```

1. Buffering data

A “specification-style” implementation of a two-place buffer can be foundon the web site in `bufferExample.scala`. As seen last week, one can also implement a two-place buffer as two connected one-place buffers. Make a copy of the `bufferExample`, and change it to use two connected one-place buffers instead. Add `System.err.println` statements at different points in the programs and study a few test runs. Do both two-place buffers behave in the same way? If not, explain the difference.

2. Scheduling

We now turn to the scheduler example that we’ve seen in class. A skeleton implementation can be found on the web site in `schedulerSkeleton.scala`. Fill in (and add if necessary) definitions for the states of each cell in the scheduler, and check that processes are scheduled according to the specification seen in class (or if they are not, what may be wrong).

3. More Scheduling

Study what happens if one process crashes (i.e., does not notify the scheduler of its termination). If you find a problem with your scheduler’s behavior in this case, modify the scheduler so that it handles this case “correctly”.