# OCCAM EXERCISE SHEET

1. A system consists of two distinct objects — `A` and `B`. `A` produces an infinite stream of numbers which `B` consumes.

   The `A` process outputs the sequence:

   *0, f(0), g(f(0)), f(g(f(0))), g(f(g(f(0)))), ...*

   where *f* returns *1000* if its argument is *0*, otherwise it doubles its argument. The operation of *g* (integer) divides its argument by *3*.

   The `B` process deals with numbers *three* at a time. For each triple, it prints them tabulated on the same line followed by their (integer) average.

   Implement this system in occam.

2. Design and implement a `differentiate` process which performs the inverse of the `integrate` process from the `demo` program. Design and implement a suitable test-rig to show its correctness.

3. The `demo` program produces an infinite amount of output. The output can be held and resumed using the normal XON/XOFF protocol obeyed by your terminals (i.e. by typing *ctl<Q>* and *ctl<S>* respectively). The program can be left by typing *ctl<C>*.

   (i) Modify the *occam* program so that typing any single character suspends output. Output is resumed by typing another character;

   (ii) Modify the program further so that the program `STOP`s if the character typed to suspend output was *Z*.

   This exercise can be done by reusing the `main` component from the `demo` program untouched — i.e. modify the function of the program by adding a (simple) process in parallel with what has already been developed; do not modify the existing components.

4. Extract the `squares` pipeline from the `demo` program and make it into a program which just outputs squares — one per line.

   Tap the intermediate channels (i.e. the output from `nos` and `integrate`) and mix their contents on to the `screen` with the normal output (i.e. each line of output tabulates one item from `nos`, one from `integrate` and one from `pairs`). *[Hint: use* `layout.max.chans` *with* `max.chans = 3`*]*

   Add a process to monitor the `keyboard`, some extra channels and whatever other processes and modifications to existing processes you deem necessary to achieve the following control over the pipeline. If the user types an:

   *S* The output stops and does not resume until another character is typed;

   *N* The `nos` process is reset to start outputting from zero again;

   *I* The `integrate` process is reset to clear its running sum to zero;

   *P* the `pairs` process changes its `plus` operation into a `minus`. Subsequent *P*s toggle `pairs` between these modes of operation;

   Any other character causes the terminal to bleep.

5. Implement the `type.ahead.buffer` (from the course slides) and introduce this in parallel between one of your answers for the previous questions (or the `slow.process` from the course slides) and the `keyboard` and `screen` channels.

6. Data consists of alternate lines of names and marks — e.g.:

   *Sue*
   *7*
   *Harry*
   *4*
   *Sue*
   *3*
   *Fred*
   *9*

   At the end of this data, there is a single blank line. Write an *occam* program to process such data and report the total mark associated with each name. For example, with the above data, the output would be:

   | *Name* | *Total Mark* |
   |---|---|
   | *Sue* | *10* |
   | *Harry* | *4* |
   | *Fred* | *9* |

   Correctly formatted data will be provided in ASCII files. Access is simply obtained by redirecting UNIX standard input.

   You will be given an *occam* process that inputs such a text stream and converts it into the protocol:

```
PROTOCOL NAME.NUMBER
  CASE
    string; INT::[]BYTE; INT
    poison
:
```

   The maximum number of different names that may occur in the data file will also be given, along with the maximum name length.

   *[This is an exercise on the use of a tagged* `PROTOCOL`.]

7. Design and implement a suitable screen display for demonstrating the ‘Dining Philosophers’ problem.

8. Design and test a digital logic circuit, `tricycle`, with two inputs `clock` and `clear` and two outputs, `bit.0` and `bit.1`, with the following behaviour:

   > Pulling the `clear` input low, sets the output bits to `01`. Then, each ‘tick’ on the clock steps the output bits through the sequence, `01, 10, 11` — which then repeats indefinitely.

   *[Hint: this can be done with 3* `jk.flip.flop`*s and some soldering.]*