INGI1131 Practical Exercises
Lab 2 extra: for hard rock programmers

Your own library of functions

We saw during last lab session that functors can be used to write executable programs. But they also play a very important role in software development. You can build a library of functions and pack them as a functor that can be imported and used by other programs. For instance, the following functor offer a library with two functions to compute the maximum and minimum between two numbers.

```
functor
export
   Max
   Min
define
   fun {Max X Y}
      if X > Y then X
      else Y end
   end

   fun {Min X Y}
      if X < Y then X
      else Y end
   end
end
```

Assuming that the functor is saved in a file with the name `minmax.oz`, the way to compile it is the following:

```
ozc -c minmax.oz
```

Note that now we use the option `-c` instead of `-x` as in the previous lab. That compilation will generate a file called `minmax.ozf`.

Linking a library with the OPI

If you want to use the `minmax.ozf` library on the OPI, you must link it as a module with the following instruction:

```
declare
[MinMax] = {Module.link ['minmax.ozf']}
```

The module will be loaded as a record containing the *exported* functions and procedures as fields of the record. Field's names will be as the name of the functions, but starting with a lower case letter. For instance, the following example uses the `Max` function of the `minmax.ozf` module loaded with variable `MinMax`.

```
{Browse {MinMax.max 42 7}}
```

Importing a library from another functor

To import a functor from another functor, you use the *import* section specifying the path to the functor you want to use. The following example has an equivalent behaviour to our example on the OPI. Compile it and run it.

```
functor
import
   Application
   System
   MinMax at 'minmax.ozf'
define
   {System.show {MinMax.max 42 7}}
   {Application.exit 0}
end
```

Note the equivalence between the calls `MinMax.max`, `System.show` and `Application.exit`. They are all invocations of functions or procedures inside modules.

Now to the exercises

After that introduction, we ask you to implement the following:

1. From Lab02, take exercises 1.b and 2.a, and build a module that exports functions `Fact`, `Sum`. Compile the functor.

2. Implement another functor that uses your library to compute and print the sum of all factorials from 1 to 10.