A job shop scheduling formulation in OPL taken from:

Lustig and Puget: "Program does not equal Program", Interfaces 31:6, 2001, pages 29-53

```
int nbMachines = . . . ;
range Machines 1..nbMachines;
int nbJobs = . . . ;
range Jobs 1..nbJobs;
int nbTasks = . . . ;
range Tasks 1..nbTasks;

Machines resource[Jobs,Tasks] = . . . ;
int+ duration[Jobs,Tasks] = . . . ;
int totalDuration = sum(j in Jobs, t in Tasks)
          duration[j,t];

scheduleHorizon = totalDuration;
Activity task[j in Jobs, t in Tasks]
          (duration[j,t]);
Activity makespan(0);

UnaryResource tool[Machines];

minimize
  makespan.end
subject to {
  forall(j in Jobs)
    task[j, nbTasks] precedes makespan;

  forall(j in Jobs)
    forall(t in 1..nbTasks-1)
      task[j,t] precedes task[j,t + 1];

  forall(j in Jobs)
    forall(t in Tasks)
      task[j,t] requires
          tool[resource[j,t]];
};
```

The first nine lines define the input data of the problem, consisting of the number of machines, the number of jobs, and the number of tasks. We use the OPL keyword `range` to create a type to correspond to an integer range. The array `resource`, declared in Line 8, is input data consisting of the identity of the machine that is needed to do a particular task within a job. Line 9 declares the array `duration` that is the time required to execute each task within each job. Line 10 computes the maximum duration of the entire schedule, which is used in Line 12 to set the schedule horizon for OPL. Line 13 declares a decision variable `task[ j,t]` for each job `j` and task `t` that is an activity. By default, the keyword `Activity` implicitly indicates a decision variable. An activity consists of three parts—a start time, an end time, and a duration. In the declaration given here, the durations of each activity are given as data. When an activity is declared, the constraint

`task[ j,t].start + task[j,t].duration = task[ j,t].end`

is automatically included in the system. Line 14 declares an activity `makespan` of duration 0 that will be the final activity of the schedule. Line 16 declares the machines to be unary resources. A unary resource is also a decision variable, and we need to decide what activity will be assigned to that resource at any particular time.

The problem is then stated in Lines 18 through 31. Lines 18 and 19 indicate that our objective is to finish the last activity as soon as possible. Lines 21 and 22 indicate that the last task of each job should precede this final activity. Lines 24 through 26 indicate the precedence order of the activities within each job. The word `precedes` is a keyword of the language. In the case of Line 26, the constraint is internally translated to the constraint

`task[j,t].end <= task[j,t1].start`

Finally, Lines 28 through 30 indicate the requirement relationship between activities and the machines by using the `requires` keyword. The declaration of the set of requirements causes the creation of a set of disjunctive constraints. For a particular machine `m` described by a unary resource `tool[m]`, let `task[j1,t1]` and `task[j2,t2]` be two activities that require that machine `m`. In other words, suppose that the data is given such that

`resource[j1,t1] = resource[j2,t2] = m`.

Then the following disjunctive constraint, created automatically by the system, describes the fact that the two activities cannot be scheduled at the same time:

```
task[j1,t1].end <= task[j2,t2].start \/ task[j2,t2].end <= task[j1,t1].start
```

Here, the symbol "\/" means "or," and the constraint states that either `task[j1,t1]` precedes `task[j2,t2]` or vice versa.