Project 2

Due: Friday, May 3rd 11:59pm

Description

In this project you will create the Prolog backend for an imaginary webapp. The webapp will allow a user to prepare a work schedule. The user can specify what work stations exists (with the number of people needed), and the employees with their skills, and time constraints. The information will be saved as facts in a file that will be consulted with your code. You need to implement `plan/1` which will depend on these facts and will be queried to compute a schedule.

Details

You will implement `plan/1` and any predicates necessary to implement it. The predicate should work even if called with an unbound variable as its parameter. The parameter should be unified with the `plan/3` structure. The three parameters represent the schedules for the morning, evening, and night shifts – in that order. Each schedule is a list of `workstation/2` structures, with the first parameter being a workstation and the second a list of employees.

The plan must follow the following requirements. If no such plan can be constructed, `plan/1` should fail.

- Every employee must work exactly one workstation for exactly one shift.
- No work station can be worked by fewer employees than the minimum.
- No work station can be worked by more employees than the maximum.
- The schedule should not contain workstations that are idle that shift.
- No workstation should be worked by an employee that should avoid it.
- No employee should be scheduled for a shift that the employee should avoid.

Input Predicates

You may assume a file containing all related facts exists, and is consulted with your code. An example file will be provided, but you should also make more for testing. It will contain facts for the following predicates.

- `employee/1`
  - Each fact will unify the first parameter with an employee
- `workstation/3`
  - Relates a workstation (first parameter) with the minimum and maximum number of employees needed (second and third parameter).
- `workstation_idle/2`

  - Relates a workstation (first parameter) with a shift (second parameter). It represents shifts in which the workstation will not be used. A shift can be one of three constants: `morning`, `evening`, `night`.
- `avoid_workstation/2`
  - Relates an employee (first parameter) with a workstation (second parameter), and represents workstations this employee cannot work at.
- `avoid_shift/2`
  - Relates an employee (first parameter) with a shift (second parameter). It represents the shift the employee cannot work. A shift can be one of three constants: `morning`, `evening`, `night`.

Hints and Advice

- You do not need to print anything.
- The `trace/0` predicate can be used to activate tracing mode.
  - Tracing mode will allow you to see the steps of Prolog's execution.
- The `findall/3` predicate can convert disjunctive answers to a list.
  - As an example, we can get a list of employees as follows.
    - `findall(E,employee(E),Employees).`
    - The first parameter tells us what we want to collect. It can be any Prolog term, but should contain at least one variable.
    - The second parameter is a goal. It will be queried, and every time it succeeds it will backtrack to generate a new answer.
    - The third parameter is a list of collected terms, each one corresponding to an answer generated by the goal.
- If you want a predicate to be true based on the disjunction of two things, use two rules.
- The cut can be useful for avoiding duplicated code.
- Negation-as-failure can be useful when making decisions based on not having something.

What to turn in

Upload your submission as a zip archive containing the following.

- Source code (.pl file(s))
  - Source code should not require a particular IDE to compile and run.
  - Should work on the cs1 and cs2 machines
- Readme (Plain text document)
  - List the files included in the archive and their purpose
  - Explain how to compile and run your project
  - Include any other notes that the TA may need
- Write-up (Microsoft Word or pdf format)
  - How did you approach the project?
  - How did you organize the project? Why?
  - Are there any predicates that can not be used in the "reverse" direction?
    - If so, what are they and why are they not reversible?
  - What problems did you encounter?

- How did you fix them?
- What did you learn doing this project?
- If you did not complete some feature of the project, why not?
  - What unsolvable problems did you encounter?
  - How did you try to solve the problems?
  - Where do you think the solution might lay?
    - What would you do to try and solve the problem if you had more time?

Grading

The grade for this project will be out of 100, and broken down as follows.

Followed Specifications (Can overlap with other criteria)	50
The correct usage of negation-as-failure and the cut	10
Proper use of unification, especially in the head of predicates	10
Proper use of lists, list manipulation, and arithmetic	10
Correct Output	10
Did Write-up	5
Write-up: Reversibility Question	5

If you were not able to complete some part of the program discussing the problem and potential solutions in the write-up will reduce the points deducted for it. For example, suppose there is a bug in your code that sometimes allows two customers to approach the same worker, and could not figure out the problem before the due date. You can write 2-3 paragraphs in the write-up to discuss this issue. Identify the error and discuss what you have done to try to fix it/find the problem point, and discuss how you would proceed if you had more time. Overall, inform me and the TA that you know the problem exists and you seriously spend time trying to fix the problem. Normally you may lose 5 points (since it is a rare error) but with the write-up you only lose 2. These points can make a large difference if the problem is affecting a larger portion of the program.