- How did you approach the project?
    - I approached the project with minimal understanding of Prolog. To bridge this gap, I began to embody the basics of Prolog and semantics. After gaining a basic understanding, I shifted my focus to the high-level design of the program, starting with the identification and definition of core predicates. This included setting up the main predicate plan/1, which serves as the backbone of the scheduler. From there, I organized the project into functional modules: data input handling, scheduling logic, and constraints management. Each module was designed to handle specific aspects of the problem, making the code more manageable and modular.
- How did you organize the project? Why?
    - plan/1 starts by gathering all employees and workstations using findall predicates. This sets the stage by listing available resources and requirements.
    - The assign_employees_to_shifts predicate distributes employees across three shifts: morning, evening, and night. It sequentially assigns employees to shifts while managing the remaining unassigned employees through recursive calls.
    - Within each shift, assign_to_shift allocates employees to specific workstations. It filters eligible employees for each workstation based on constraints like avoiding certain workstations or shifts.
    - select_employees_for_workstation is used to ensure that each workstation has the correct number of employees, adhering to its minimum and maximum requirements. This predicate uses recursion and backtracking to explore possible groupings that satisfy all constraints.
    - valid_plan checks the overall validity of the constructed schedule by ensuring each shift meets the workstation staffing requirements without any violations.
- Are there any predicates that can not be used in the “reverse” direction? If so, what are they and why are they not reversible?
    - The predicates that are not reversible in this project primarily include those involving generated data like findall/3. These predicates do not work in reverse because they produce output that does not readily translate back into input queries. For example, reversing findall(E, employee(E), Employees) would require inferring individual employee/1 facts from a list of employees, which Prolog is not designed to do directly.
- What problems did you encounter?
    - During the project, I encountered several issues including stack overflow errors, unknown procedure errors, and unexpected behaviors in code execution. These problems often stemmed from the intricacies of Prolog’s logic programming model, which was different from other programming paradigms I was familiar with.
- How did you fix them?
    - To address these issues, I adopted a systematic approach to debugging that involved checking each predicate for logical consistency and ensuring that all clauses were correctly defined and called. I used Prolog's tracing facilities to step

through the execution process, which helped identify where the logic did not behave as expected or where the recursion might be causing stack overflows.

- What did you learn doing this project?
    - This project was a significant learning experience where I transitioned not only between programming paradigms but also shifted my thinking to a more logical framework, typical of Prolog's environment. I learned the importance of understanding the flow of logic in programming and gained valuable skills in debugging and structuring complex programs in a logical manner. These skills will be beneficial in future programming endeavors across various languages and paradigms.
- If you did not complete some feature of the project, why not?
    - One aspect I struggled with was adequately mastering Prolog's debugging tools and techniques. Unlike Haskell, which I was more familiar with, Prolog allows for side effects that I was not initially aware of. This gap in understanding impacted my ability to debug effectively, preventing me from completing some features as planned. I was unable to fix some of the issues I was facing and was not prepared with the prefacing knowledge to solve these issues.
    - What unsolvable problems did you encounter?
        - A persistent issue was receiving 'false' for certain queries, indicating failure in meeting the conditions or reaching a valid solution within the provided constraints.
    - How did you try to solve the problems?
        - I attempted to resolve these issues by revising and testing individual predicates, trying to isolate the conditions under which they failed. This involved modifying and refining the logic used in various parts of the scheduling algorithm.
    - Where do you think the solution might lay?
        - I suspect that the solution lies in a deeper understanding of each predicate's role and the interactions between them. Clearer insight into how predicates influence each other could reveal where the logic fails.
        - What would you do to try and solve the problem if you had more time?
            - Given more time, I would methodically dissect each predicate, examining its logical structure and its impact on the overall program. I would focus on understanding how each piece of the program fits into the larger puzzle, rather than jumping straight into debugging. This approach would likely provide a clearer roadmap for identifying and fixing errors more efficiently. I would also spend significantly more time learning about prolog and ALL of its capabilities
-