# Mini Expert System: Prolog Student Course Management System Documentation

1. **Knowledge Base**

The knowledge base is the foundational component of the expert system. It contains all the necessary information that the system uses to make decisions and provide recommendations.

## a. Courses

- The `course/3` predicate defines each course offered within the system.
- **Structure:** `course(CourseID, CourseName, Credits).`
  - **Example:** `course(cs101, 'Introduction to Computer Science', 4).`
- **Details:**
  - `CourseID`: A unique identifier for each course.
  - `CourseName`: The name of the course.
  - `Credits`: The number of credit hours the course is worth.

## b. Prerequisites

- The `prerequisite/2` predicate specifies the prerequisite relationships between courses.
- **Structure:** `prerequisite(Course, PrerequisiteCourse).`
  - **Example:** `prerequisite(cs102, cs101).`
- **Details:**
  - `Course`: The course that requires a prerequisite.
  - `PrerequisiteCourse`: The course that must be completed before enrolling in the associated course.

## c. Majors and Requirements

- The `major/2` predicate defines which courses are required for each major.
- **Structure:** `major(MajorName, CourseID).`
  - **Example:** `major(cs, cs101).`
- **Details:**
  - `MajorName`: The name of the major (e.g., Computer Science, Mathematics).
  - `CourseID`: A course required to complete the major.

## d. Students

- The `student/5` predicate stores information about each student.
- **Structure:** `student(StudentID, Name, Batch, Department, Major).`
  - **Example:** `student(12345, 'John Doe', 2023, cs, cs).`
- **Details:**
  - `StudentID`: A unique identifier for the student.
  - `Name`: The student's full name.
  - `Batch`: The year the student joined.
  - `Department`: The department the student belongs to.
  - `Major`: The student's major.

## e. Completed Courses

- The `completed/2` predicate tracks which courses a student has finished.
- **Structure:** `completed(StudentID, CourseID).`
  - **Example:** `completed(12345, cs101).`
- **Details:**
  - `StudentID`: The unique identifier for the student.
  - `CourseID`: The course that the student has completed.

## f. Performance

- The `performance/3` predicate (though not shown in the provided snippet, it's declared as dynamic) would likely track the grades or performance of students in specific courses.
- **Structure:** `performance(StudentID, CourseID, Grade).`
  - **Example:** `performance(12345, cs101, 85).`
- **Details:**
  - `StudentID`: The unique identifier for the student.
  - `CourseID`: The course in which the grade was achieved.
  - `Grade`: The grade the student received.

## 2. Inference Engine

The inference engine applies logical rules to the knowledge base to draw conclusions and make decisions.

## a. Eligibility to Take a Course

- The `can_take/2` predicate determines if a student is eligible to enroll in a particular course.
- **Structure:** `can_take(StudentID, CourseID).`
  - **Example:**

```
can_take(StudentID, Course) :-
    course(Course, _, _),  % The course must exist.
    student(StudentID, _, _, _, Major),  % Get the stud
ent's major.
    major(Major, Course),  % The course must be require
d for the student's major.
    \\+ completed(StudentID, Course),  % The course mus
t not be completed yet.
    \\+ has_unmet_prereq(StudentID, Course).  % The stu
dent must have met all prerequisites.
```

- **Details:**

- Checks if the course exists.
- Verifies the course is part of the student's major requirements.
- Ensures the student hasn't completed the course.
- Confirms all prerequisites for the course are met.

## b. Checking Prerequisites

- The `has_unmet_prereq/2` predicate checks if a student has unmet prerequisites for a course.
- **Structure:** `has_unmet_prereq(StudentID, CourseID).`
  - **Example:**

```
has_unmet_prereq(StudentID, Course) :-
    prerequisite(Course, Prereq),  % Identify the prere
quisite for the course.
    \\+ completed(StudentID, Prereq).  % Check if the p
rerequisite is completed.
```

- **Details:**
  - Identifies any prerequisite for a course.
  - Verifies whether the student has completed the prerequisite course.

## 3. Knowledge Acquisition and Learning Module

The knowledge acquisition module is responsible for adding new knowledge to the system or modifying existing knowledge. This is facilitated by the use of dynamic predicates, which allow for runtime updates.

## a. Adding New Information

- New courses, students, and completion records can be added dynamically using the `assert/1` predicate.
  - **Example:**

```
assert(course(cs104, 'Advanced Programming', 4)).
assert(student(67890, 'Jane Smith', 2022, cs, cs)).
assert(completed(67890, cs101)).
```

### b. Modifying or Removing Information

- Existing knowledge can be retracted using the `retract/1` predicate.
  - **Example:**

```
retract(course(cs104, 'Advanced Programming', 4)).
```

## 4. User Interface

The user interface in this Prolog-based expert system is typically the Prolog interpreter itself, where users can query the system to get advice or perform actions.

### a. Queries

- Users can query the system for advice using predicates like `can_take/2`.
  - **Example:** To check if a student can take `cs102`:

```
can_take(12345, cs102).
```

### b. Interactive Features

- Users can interact with the system by entering new facts or rules, querying the knowledge base, and getting explanations for decisions.

## 5. Explanation Module

The explanation module provides justifications for the decisions made by the inference engine. It could explain why a student can or cannot take a course, based on the rules and knowledge base.

### a. Explanation of Course Eligibility

- If a student cannot take a course, the system could explain which prerequisites are unmet or if the course isn't part of the major requirements.
    - **Example Query:**

      ```
      ?- can_take(12345, cs102).
      ```

    - **Explanation:**
      If the student cannot take `cs102`, the system might explain:
      "Student 12345 cannot take cs102 because they have not completed the prerequisite cs101."

## b. Detailed Reasoning

- The system could be extended to provide detailed explanations, tracing the logic of decisions, and helping users understand the underlying rules.

## Conclusion

This Prolog-based expert system is structured around the five key components: Knowledge Base, Inference Engine, Knowledge Acquisition and Learning Module, User Interface, and Explanation Module. Each component plays a crucial role in advising students on course selection, ensuring they meet prerequisites, and providing detailed explanations of the advice given. The system is designed to be dynamic and adaptable, allowing it to grow and change as new knowledge is added.