Programming questions must be answered in Prolog, with programs that show good declarative style. Marks will be deducted for programs that look like translated Java or C. Simple programs are preferred, especially small collections of simple rules. You should turn in four files: a file of your solutions to the non-programming questions, a source listing of all your Prolog code, a transcript of a terminal session with the Prolog interpreter, and a signed cover sheet. The terminal session is very important. It must show both the input provided by you and the output printed by the Prolog interpreter. The terminal session should be short and should demonstrate that your programs and queries work correctly. Readability and well-written documentation are of great importance. The marker may run your code if he has any doubts about its correctness. For full marks, all answers returned by Prolog must be correct; it is not sufficient for just the first answer to be correct. Your terminal session must show all the answers.

In this assignment, non-logical operations are rarely needed, and their use should be minimized. In particular, do not use functor, arg, =.., assert, retract, fail or cut. However, the built-in predicates atom and number, as well as negation and arithmetic operations, may be needed from time to time, and you may use them when necessary. You may also use the underscore character, _, for anonymous variables. Do not use the equality predicate, =, unless it reduces the size of your rules. Finally, do not use ; (semicolon), which stands for logical “or” in Prolog.

The marker has a limited amount of time for each assignment, so it is your responsibility to provide documentation and testing that allows him to quickly evaluate your work. As with all work in this course, 20% of the grade is for quality of presentation.

No more questions will be added
1. **Unification** (18 points total).

For each of the following pairs, state whether or not the atomic formulas unify. In each case, state what the most general unifier is, or show that unification is impossible (2 points each). In this problem, $U, V, X, Y, Z$ are variables, and $a, b, c, d$ are constant symbols.

(a) $p(a, X)$ and $q(Y, b)$
(b) $p(X, X)$ and $q(U, V)$
(c) $q(a, b, X)$ and $q(Y, Z, Y)$
(d) $p(c, Y, d, Y)$ and $p(U, U, V, V)$
(e) $p(g(a, X))$ and $p(g(U, b))$
(f) $p(f(a, b), f(X, Y))$ and $p(U, U)$
(g) $p(Y, f(a, Y))$ and $p(g(c, d), U)$
(h) $p(g(X, Y), h(Z, Z))$ and $p(U, h(U, a))$
(i) $p(g(X, Y), g(Y, Z))$ and $p(U, U)$

2. **First-Order Logic** (27 points total).

This question uses the following predicates:

- $boat(X)$ means that $X$ is a boat.
- $piece(X)$ means that $X$ is a chess piece.
- $colour(X, C)$ means that object $X$ has colour $C$.
- $type(X, Y)$ means that object $X$ has type $Y$. Here $Y$ can take on values such as *king*, *queen*, *pawn*, *canoe*, *yacht*, *catamaran*, etc.

Using only these predicates, translate each of the English statements below into first-order logic (3 points each). You may use the logical operators $\forall$, $\exists$, $\land$, $\lor$, $\neg$, $\rightarrow$, $\leftarrow$, $\leftrightarrow$ and the predicates above. Do not use any function symbols or arithmetic symbols, and use as few constant symbols as possible. You may use the equality predicate if necessary. You may also use arbitrary formulas of first-order logic, not just Horn rules. (This is a paper-and-pencil problem, and no programming is required.)

(a) Every chess piece is black or white.
(b) No chess piece is red.
(c) No chess piece is both black and white.
(d) A boat is not a chess piece.
(e) Every yacht is a boat.
(f) No chess piece is a canoe.
(g) At least one chess piece is a black queen.
(h) No more than one chess piece is a black queen.
(i) Exactly two chess pieces are kings.
3. **Database Construction** (5 points).

Create a Prolog database for a university. (This database will be used in the next two questions.) The database consists of ground atomic formulas made from the following predicates:

```prolog
student(name, gpa, dept)
professor(name, salary, dept)
course(course#, year, prof_name)
enrolment(course#, year, student_name)
```

These predicates are interpreted as follows. Each student has a name, grade point average, and department. Each professor has a name, salary and department. Each course has a course number, year and a professor who teaches it. Finally, the enrolment predicate describes which students take which courses in each year. A course may have many students, a student may take many courses, a professor may teach many courses, but each course may be offered only once a year.

Here are typical atomic formulas for each of the four predicates:

```prolog
student(einstein, 3.9, physics)
professor(newton, 85000, physics)
course(ph250, 1998, newton)
enrolment(ph250, 1998, einstein)
```

Your database should contain at least 40 atomic formulas using the four predicates above. Choose the data so that the queries in Questions 4 and 5 return interesting non-trivial answers (e.g., not always an empty answer, not always the same answer, not always a single answer).

4. **Database Queries** (14 points total).

Without adding any rules to your Prolog database, pose queries to Prolog that retrieve the the information below (and no more). You may use inequality predicates; e.g., $x < y$ is true iff $x$ is less than $y$.

(a) (2 points) The name and gpa of all students in Physics.

(b) (2 points) The course numbers of all courses taught by Leonard Hofstadter.

(c) (3 points) The course number and year of all courses taken by Sheldon Cooper that were taught by Stephen Hawking.

(d) (3 points) The names of all students who took phy450 from Richard Feynman and the years in which they took it.

(e) (4 points) The name and gpa of all students who took csc324 from Allen Turing and the years in which they took it.
Note: your solutions must be independent of the data stored in the database, as described below. You should use the Prolog predicate `setof` to eliminate duplicate answers.

5. **Deductive Databases (16 points total).**

Add rules to your Prolog database that define the predicates below (?? points each). Test your rules by posing queries to the database you constructed in Question 3. Of course, your rules should work for any reasonable set of data, not just the particular data in your database. For example, if a new student enrolls in a course, or if a department hires a new professor, then your rules should still work without modification. This property of a rulebase is called *data independence*. You should also use the Prolog `setof` predicate to eliminate duplicate answers.

(a) (2 points) `hotshot(s)`: True if student `s` has a gpa greater than 3.8.
(b) (3 points) `taught(p,s)`: True if professor `p` has ever taught a course taken by student `s`.
(c) (3 points) `offered(c,y,d)`: True if course `c` was taught in year `y` by a professor in department `d`.
(d) (4 points) `taken(d1,d2)`: True if a student in department `d1` has ever taken a course offered by department `d2`.
(e) (4 points) `neverTaken(d1,d2)`: True if no student in department `d1` has ever taken a course offered by department `d2`. Use this predicate to retrieve all those departments that have a student who has never taken a course offered by the Math department. Likewise, retrieve all those departments that have never offered a course taken by a student in the English department.

6. **Recursion in Deductive Databases (23 points total).**

In this question, you will define a family tree and pose queries to it using rules. The tree itself will be defined in terms of three predicates: `parent`, `male` and `female`. Intuitively, `parent(X,Y)` means that person `X` is a parent of person `Y`. For instance, the formulas `parent(patricia,anna)` and `parent(patricia,martin)` mean that Patricia is a parent of Anna and also of Martin. Likewise, `male(X)` means that person `X` is male, and `female(X)` means that person `X` is female. As in the previous question, all your Prolog rules should be data independent.

(a) (3 points) Use the predicates `male`, `female` and `parent` to build a database of atomic formulas that represents a small family tree spanning four generations of Roman nobility. The people at the bottom of the tree are thus the great-grand-children of the people at the top of the tree. Don’t worry about historical accuracy, but to give the marker some common references, put the names Romulus and Remus in the first generation (the top of the tree); put the names Cicero and Scipio in the second generation; put the names Julius, Augustus, and Livia in the third generation; and put the names Caligula, Drussus, Nero, Germanicus, and Claudius in the fourth generation (the bottom of the tree).
Fill out the family tree with at least ten other names. Be sure that each person in the tree has at most one mother (female parent) and one father (male parent). In addition, to represent inbreeding amongst the nobility, your family “tree” must be a directed acyclic graph (DAG). You should construct this graph so that the queries in the rest of this question test your program. The queries should return interesting and non-trivial answers.

(b) (12 points total) Write Prolog rules that define the predicates below (3 points each). Test each of the predicates by posing queries to your Roman database. (e.g., the query grandchild(X,cicero) retrieves the grandchildren of Cicero.) You should also test the queries in reverse. (e.g., the query grandchild(nero,Y) retrieves the grandparents of Nero. Use the setof predicate to eliminate duplicate answers.

\begin{align*}
\text{brother}(X,Y), & \quad \text{which means that } X \text{ is a brother (or half brother) of } Y. \\
\text{cousin1}(X,Y), & \quad \text{which means that } X \text{ is a first cousin of } Y \text{ (i.e., they have a common grandparent).} \\
\text{cousin2}(X,Y), & \quad \text{which means that } X \text{ is a second cousin of } Y \text{ (i.e., they have a common great grandparent). Define } \text{cousin2} \text{ in terms of } \text{cousin1.} \\
\text{grandchild}(X,Y), & \quad \text{which means that } X \text{ is a grandchild of } Y.
\end{align*}

(c) (8 points total) Write a recursive Prolog program that defines a predicate descendant\((X,Y)\), which means that person \(X\) is a descendant of person \(Y\) (4 points). This program should work for any family tree, even if it has a thousand generations. Using the descendant predicate, pose queries to your Roman database that retrieve the data below (1 point each). Eliminate duplicate answers.

i. The descendants of Cicero.

ii. The descendants of Livia.

iii. Those people who are both descendants of Cicero and descendants of Livia.

iv. The first cousins of the descendants of Scipio.

7. **Function Terms and Recursion** (30 points total).

In this question, you will use function terms to represent trees. Each internal node of the tree is either red or black. A red node has two children, a black node stores three children, and every node stores a number. The function term \(\text{red}(N,C_1,C_2)\) represents a red node that stores number \(N\) and whose children are \(C_1\) and \(C_2\). Likewise, the function term \(\text{black}(N,C_1,C_2,C_3)\) represents a black node that stores number \(N\) and whose children are \(C_1, C_2\) and \(C_3\). Finally, the function term \(\text{leaf}(N)\) represents a leaf node that stores the number \(N\). For example, the function term \(\text{red}(3,\text{leaf}(2),\text{black}(6,\text{leaf}(1),\text{leaf}(4),\text{leaf}(2)))\) represents a simple red-black tree.

(a) (10 points) Define a predicate \(\text{treeSum}(T,N)\) that is true if \(N\) is the sum of all the numbers in tree \(T\). For example,
(b) (10 points) Define a predicate \texttt{incRed(T1,T2)} that is true if T2 is the result of adding 1 to all the red nodes in tree T1.

\begin{verbatim}
?- incRed(black(1,leaf(2),red(3,leaf(4),leaf(5)),leaf(6)),T).
  T = black(1,leaf(2),red(4,leaf(4),leaf(5)),leaf(6))
\end{verbatim}

(c) (10 points) Define a predicate \texttt{leafList(T,L)} that is true if L is a list of all the numbers stored in the leaves of tree T.

\begin{verbatim}
?- leafList(black(1,leaf(2),red(3,leaf(4),leaf(5)),leaf(6)),L).
  L = [2,4,5,6]
\end{verbatim}

8. **Nested Lists** (15 points).

Redo Question 5 of Assignment 1 in Prolog. That is, define a predicate \texttt{leftmost(X,Y)} that is true if Y is the leftmost symbol in nested list X. For example,

\begin{verbatim}
?- leftmost([1,[2,[3,b],c],d],Y).
  Y = b
\end{verbatim}

If X contains no symbols, then \texttt{leftmost(X,Y)} should either return \texttt{false} or return \texttt{Y=[]}, as long as it always does one or the other. You may use the Prolog predicates \texttt{number} and \texttt{atom}. (Note that \texttt{atom([])} is true.)
Complete this page and submit it with your assignment.

Name: ____________________________  (Underline your last name)

Student number: __________________________

I declare that this assignment is solely my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters.

Signature: ____________________________