CS 603 Exam 3 **Spring 2007** **Solution**

1. Assume the Prolog database defines relations of the form male(X), female(X), sibling(X, Y), spouse(X, Y), and parent(X, Y), which means that X is a parent of Y. Write the Prolog predicates described below.

   a. niece(X, Y), which means that X is a niece of Y. **[5 points]**
   Note: your niece is your brother’s daughter, sister’s daughter, or spouse’s niece.

   ```
   niece(X, Y) :– female(X), parent(Z, X), sibling(Z, Y).
   niece(X, Y) :– female(X), parent(Z, X), sibling(Z, W), spouse(W, Y).
   ```

   b. nephew(X, Y), which means that X is a nephew of Y. **[5 points]**
   Note: your nephew is your brother’s son, sister’s son, or spouse’s nephew.

   ```
   nephew(X, Y) :– male(X), parent(Z, X), sibling(Z, Y).
   nephew(X, Y) :– male(X), parent(Z, X), sibling(Z, W), spouse(W, Y).
   ```

   c. uncle(X, Y), which means that X is an uncle of Y. **[5 points]**
   Note: your uncle is your father’s brother, mother’s brother, or aunt’s husband.

   ```
   uncle(X, Y) :– male(X), sibling(X, Z), parent(Z, Y).
   uncle(X, Y) :–  male(X), spouse(X, W), sibling(W, Z), parent(Z, Y).
   ```

   d. aunt(X, Y), which means that X is an aunt of Y. **[5 points]**
   Note: your aunt is your father’s sister, mother’s sister, or uncle’s wife.

   ```
   aunt(X, Y) :– female(X), sibling(X, Z), parent(Z, Y).
   aunt(X, Y) :– female(X), spouse(X, W), sibling(W, Z), parent(Z, Y).
   ```

   e. cognate(X, Y), which means that X is blood-related to Y. **[6 points]**

   ```
   cognate(X, X).
   cognate(X, Y) :– parent(Z, X), cognate(Z, Y).
   cognate(X, Y) :– parent(Z, Y), cognate(Z, X).
   ```

2. Write the following predicate using Prolog: prime(N) succeeds if N is a prime number. Examples: prime(11) succeeds, but prime(12) fails. **[8 points]**

```
prime(N) :- N > 1, M is N - 1, check(N, M).

check(_, 1).
check(N, M) :- M > 1, R is N - (M*(N/M)), R > 0, P is M - 1, check(N, P).
```

3. Write the following predicate using Prolog: selectionsort(X, Y) succeeds if Y is the list that is obtained by sorting the elements of list X into ascending order. Your code must use the selection sort algorithm, which at each step locates the smallest remaining value and places it at its correct position in the list. Do <u>not</u> use any other sorting algorithm such as insertion sort, merge sort, or quick sort. Example: selectionsort([9, 5, 2, 7, 4, 0], L) succeeds with L = [0, 2, 4, 5, 7, 9]. **[12 points]**

```
selectionsort([ ], [ ]).
selectionsort(L, [H|T]) :- minimum(L,H), delete(H,L,X), selectionsort(X,T).

minimum([X], X).
minimum([X,Y|Z], M) :- X<Y, minimum([X|Z], M).
minimum([X,Y|Z], M) :- X>=Y, minimum([Y|Z], M).

delete(H, [H|T], T).
delete(X, [H|T], [H|U]) :- X<H, delete(X,T,U).
delete(X, [H|T], [H|U]) :- X>H, delete(X,T,U).
```

4. Suppose you are given the following logic program:

```
← a(1, 2).
← b(3, 4).
a(X, Y), c(X, Y) ←.
b(X, Y), d(Y, X) ←.
a(T, U), b(V, W), e(U, T, W, V) ← c(Q, S), d(P, R), e(P, Q, R, S).
← f(5).
← g(6).
h(7) ←.
j(8) ←.
f(U), g(V) ← k(U, W), m(W, V).
k(Z, X), m(Y, Z) ← h(X), j(Y).
```

a. Use resolution to determine a conclusion that involves only the e relation. **[8 points]**

c(1, 2) ←.
d(4, 3) ←.
e(2, 1, 4, 3) ← e(4, 1, 3, 2).

b. Use resolution to determine a conclusion that involves only the k relation. **[5 points]**

← k(5, W), m(W, 6).
k(Z, 7), m(8, Z) ←.
k(6, 7) ← k(5, 8).

c. Use resolution to determine a conclusion that involves only the m relation. **[5 points]**

← k(5, W), m(W, 6).
k(Z, 7), m(8, Z) ←.
m(8, 5) ← m(7, 6).

5. Rewrite each logical expression as an equivalent completely simplified expression.

   a. $(a \vee \neg b) \wedge (\neg a \vee b)$ **[3 points]**

      $\mathbf{a \leftrightarrow b}$

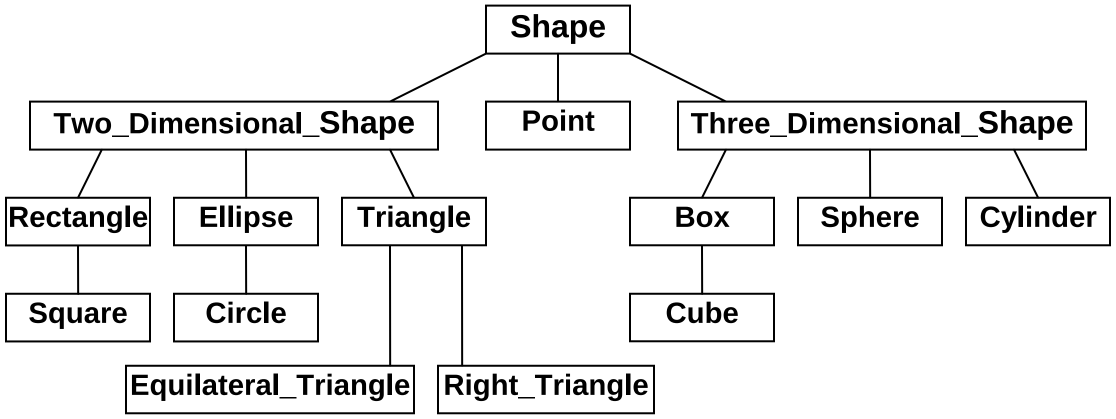
   b. $\neg(a \rightarrow (b \rightarrow c))$ **[3 points]**

      $\mathbf{a \wedge b \wedge \neg c}$

   c. $(\forall x)(P(x) \rightarrow Q(x)) \wedge (\forall y)(Q(y) \rightarrow \neg R(y)) \wedge \neg(\exists z)(\neg R(z))$ **[5 points]**

      $\mathbf{(\forall x)(\neg P(x) \wedge \neg Q(x) \wedge R(x))}$

6. Draw an object-oriented class inheritance hierarchy that satisfies the "is-a" relationship (or "principle of substitutability") and that includes these classes: Square, Rectangle, Cube, Box, Shape, Two_Dimensional_Shape, Three_Dimensional_Shape, Circle, Ellipse, Point, Right_Triangle, Equilateral_Triangle, Triangle, Sphere, and Cylinder. **[10 points]**



7. Write these classes from the previous problem using μSmalltalk: Square, Rectangle, Cube, Box, Shape, Two_Dimensional_Shape, and Three_Dimensional_Shape. Utilize the class inheritance hierarchy appropriately to reuse code everywhere possible. For maximum credit, each class should have at most two instance variables. (Hint: instance variables are not necessarily always integers.) The clients traced below illustrate how some of the classes should behave. Note: the perimeter of a box or cube is the sum of all its edge lengths, and the area of a box or cube denotes its surface area. **[35 points]**

```
-> (val r (new:: Rectangle 3 4))
-> (dimensions r)    ; returns 2
-> (perimeter r)     ; returns 14
-> (area r)          ; returns 12
-> (volume r)        ; returns 0
-> (display r)
Dimensions=2
Perimeter=14
Area=12
Volume=0
```

```
-> (val b (new::: Box 3 4 5))
-> (dimensions b)    ; returns 3
-> (perimeter b)     ; returns 48
-> (area b)          ; returns 94
-> (volume b)        ; returns 60
-> (display b)
Dimensions=3
Perimeter=48
Area=94
Volume=60
```

```
-> (val s (new: Square 5))
-> (dimensions s)    ; returns 2
-> (perimeter s)     ; returns 20
-> (area s)          ; returns 25
-> (volume s)        ; returns 0
-> (display s)
Dimensions=2
Perimeter=20
Area=25
Volume=0
```

```
-> (val c (new: Cube 10))
-> (dimensions c)    ; returns 3
-> (perimeter c)     ; returns 120
-> (area c)          ; returns 600
-> (volume c)        ; returns 1000
-> (display c)
Dimensions=3
Perimeter=120
Area=600
Volume=1000
```

```
(class Shape Object ( )
        (method display ( )
                (print #Dimensions=) (println (dimensions self))
                (print #Perimeter=) (println (perimeter self))
                (print #Area=) (println (area self))
                (print #Volume=) (println (volume self))
                nil)
)
(class Two_Dimensional_Shape Shape ( )
        (method dimensions ( ) 2)
        (method volume ( ) 0)
)
(class Rectangle Two_Dimensional_Shape (width length)
        (classMethod new:: (W L)
                (init:: (new Rectangle) W L))
        (method init:: (W L)
                (set width W)
                (set length L)
                self)
        (method perimeter ( ) (* 2 (+ width length)))
        (method area ( ) (* width length))
)
(class Square Rectangle ( )
        (classMethod new: (S)
                (init:: (new Square) S S))
)
(class Three_Dimensional_Shape Shape ( )
        (method dimensions ( ) 3)
)
(class Box Three_Dimensional_Shape (base height)
        (classMethod new::: (W L H)
                (init::: (new Box) W L H))
        (method init::: (W L H)
                (set base (new:: Rectangle W L))
                (set height H)
                self)
        (method perimeter ( )
                (+ (* 2 (perimeter base)) (* 4 height)))
        (method area ( )
                (+ (* 2 (area base)) (* (perimeter base) height)))
        (method volume ( ) (* (area base) height))
)
(class Cube Box ( )
        (classMethod new: (S)
                (init::: (new Cube) S S S))
)
```