Homework #2

1. The following C function computes for the power ab where a is a floating point and b is a (non-negative) integer.

```
double power(double a, int b)
{
        Int i;
        double temp = 1.0;
        for (i = 0; i <= b; i++)
                temp *= a:
        return temp;
}
```

   a. Rewrite the procedure in a functional form.
   b. Rewrite your answer to (a) using an accumulating parameter to make it tail recursive.

2.

The binomial coefficients are a frequent computational task in computer science. They are defined as follows for $n >= 0$, $0 <= k <= n$ (! is factorial and $0! = 1$):

$$B(n, k) = \frac{n!}{(n - k)!\, k!}$$

(a) Write a procedure using a loop to compute $B(n, k)$. Test your program on $B(10, 5)$.
(b) Use the following recurrence and the fact that $B(n, 0) = 1$ and $B(n, n) = 1$ to write a functional procedure to compute $B(n, k)$:

$$B(n, k) = B(n - 1, k - 1) + B(n - 1, k)$$

Test your program on $B(10, 5)$.

3.

A **refutation system** is a logical system that proves a statement by assuming it is false and deriving a contradiction. Show that Horn clause logic with resolution is a refutation system. (*Hint*: The empty clause is assumed to be false, so a goal $\leftarrow a$ is equivalent to $a \rightarrow$ false. Show that this is equivalent to not(a).)

4.

Write the following statements in the first-order predicate calculus:
If it is raining or snowing, then there is precipitation.
If it is freezing and there is precipitation, then it is snowing.
If it is not freezing and there is precipitation, then it is raining.
It is snowing.

5.

Show that the following grammar does not satisfy the second rule of predictive parsing:

stmt → *if-stmt* | `other`
if-stmt → `if` *stmt* [`else` *stmt*]

6.

Given the following grammar in EBNF:

expr → (*list*) | `a`
list → *expr* [*list*]

(a) Show that the two conditions for predictive parsing are satisfied.
(b) Write a recursive-descent recognizer for the language.

7.

Given the following BNF:

expr → (*list*) | `a`
list → *list* , *expr* | *expr*

(a) Write EBNF rules and/or syntax diagrams for the language.
(b) Draw the parse tree for `((a, a), a, (a))`.
(c) Write a recursive-descent recognizer for the language.