As described in lecture 6, write a Scheme function called `sat` that takes a single argument, a list of integers representing a SAT problem in the DIMACS format. Your function should return `#t` if the formula is satisfiable and `#f` if the formula is not satisfiable.

Example:

```scheme
> (sat '(1 0 -1 0))
#f
> (sat '(1 2 0 -1 -2 0))
#t
> (sat '(-1 -3 0 -1 -5 0 -3 -5 0 -2 -4 0 -2 -6 0 -4 -6 0 1 2 0 3 4 0 5 6 0))
#f
```

Here are some suggestions to get you started:

- These are the functions we wrote in class to get the number of clauses and maximum variable from a SAT problem:

  ```scheme
  (define (countclauses data)
    (cond
      ((null? data) 0)
      ((= (car data) 0) (+ 1 (countclauses (cdr data))))
      (#t (countclauses (cdr data))))
  
  (define (maxvar data)
    (cond
      ((null? data) 0)
      (#t (let* ((max (maxvar (cdr data)))
                (c (car data))
                (a (if (< c 0) (- c) c)))
               (if (> a max) a max))))
  ```

- Write a function `getClauses` that takes a single argument, a list of integers representing a SAT problem in DIMACS format, and returns a list of lists, where each list represents a clause.

Example:

```scheme
> (getClauses '(1 2 0 -1 -2 0))
((1 2) (-1 -2))
> (getClauses '(-1 -3 0 -1 -5 0 -3 -5 0 -2 -4 0 -2 -6 0 -4 -6 0 1 2 0 3 4 0 5 6 0))
((-1 -3) (-1 -5) (-3 -5) (-2 -4) (-2 -6) (-4 -6) (1 2) (3 4) (5 6))
```
A truth assignment is a function from variables to the set \{T, F\}. If the number of variables is \(n\), one way to represent a truth assignment is with a list of length \(n\) in which the \(i^{th}\) element represents the value for variable \(i\). For example, \((#t #f #f)\) represents an assignment for three variables in which the first is true and the others are false.

Write a function `allassign` that takes a positive integer \(n\) and generates all possible assignments of length \(n\).

Example:

```plaintext
> (allassign 2)
((#f #f) (#f #t) (#t #f) (#t #t))
```

- Write a function `eval-var` that takes a variable (a positive integer) and a list representing an assignment and returns the value of that variable under that assignment.

- Write similar functions `eval-lit` to evaluate a literal, `eval-clause` to evaluate a clause, and `eval-form` to evaluate a CNF formula.

- The satisfiability problem can be solved by evaluating the formula for each possible assignment and if any of them are true, returning true.

A data file with increasingly difficult problems (all of which are unsatisfiable) is posted on the web site with the assignment. How many can you solve?