This is a Prolog programming assignment. You are expected to write a Prolog program that consists of predicates described below. The predicates are unrelated to each other, and cover a variety of Prolog programs that are similar to those we have gone through in the class. You will place the definitions of all the predicates in a single file and submit it via Blackboard (see submission instructions at the end of this handout).

1. **interleave:** Write a predicate interleave(L1, L2, L) such that, given two lists L1 and L2, it returns a list L that is an interleaving of L1 and L2. An interleaving merges the two lists, maintaining the order of elements in the input lists. For instance, interleave([1,2], [a,b], L) will return L=[1,2,a,b] and, upon backtracking, return L=[1,a,2,b], L=[1,a,b,2], L=[a,1,2,b], L=[a,1,b,2], and L=[a,b,1,2] (the answers themselves may be in any order).
2. **genprimes:** Write a predicate genprimes(N, L) that, given a positive integer N returns the list of all prime numbers between 2 and N (inclusive).
3. **flatten:** Write a predicate flatten that flattens a given list, returning a list of integers in the same order as they appear in the given list. For instance:
   * flatten([1,2,3], L2) succeeds with L2 = [1,2,3]
   * flatten([], L2) succeeds with L2 = []
   * flatten([1, [5,6], 2,4], L2) succeeds with L2 = [1,5,6,2,4]
   * flatten([[[1,2],[3,4]],5,[6,[7,8]]], L2) succeeds with L2 = [1,2,3,4,5,6,7,8].

For full credit, flatten should run in time linear in the size of the given list (first argument). **Hint:** use difference lists.

1. **knapsack:** Let item(ID, Size) be a set of facts representing objects (by some unique ID) and their sizes (in some universal unit). We want to select from the set of all available objects a subset whose total size does not exceed a given threshhold. Write a predicate knapsack(N, L) to implement this, where N is a positive integer that represents the given threshold. knapsack returns in L some bag of items, sum of whose sizes does not exceed N. Note that L is a bag: the same item may appear multiple times. For instance, if item facts are as follows:

item(a, 5).

item(b, 3).

item(c, 7).

knapsack(10, L) returns the following answers (upon backtracking):

L=[a]

L=[a,a]

L=[a,b]

L=[b]

L=[b,a]

L=[b,b]

L=[b,b,b]

L=[b,c]

L=[c]

1. **eval:** Consider the language of integer arithmetic expressions given by the following grammar:

E -> T '+' E

E -> T

T -> F '\*' T

T -> F

F -> '(' E ')'

F -> integer

Let strings be represented by lists. For instance, the expression 2+3\*5 is written as [10, '+', 4, '\*', 8]. Write a predicate eval(S, V) that, given a string in list form in S:

* + fails if S does not represent a valid expression (according to the above grammar), and
  + succeeds binding V to the value of the expression otherwise.

For example, eval([10,'+',4,'\*'], V) fails, while eval([10,'+',4,'\*',8],V) succeeds with V=42.

1. **sat:** Consider propositional boolean formulae made up of boolean constants true and false, binary boolean operators and and or, unary boolean operator not, and propositional variables. For this problem, we will represent propositional variables by Prolog variables, and propositional boolean formulae by Prolog terms.

For instance,

* + and(P, true) is a term representing propositional formula *P ∧ true*.
  + or(P, not(P)) is a term representing propositional formula *P ∨ ¬ P*.
  + and(P, or(Q, not(P))) is a term representing propositional formula *P ∧ (Q ∨ ¬ P)*.

Write a predicate sat(F), where F is a term representing a propositional boolean formula, that

* + succeeds, binding the propositional variables in F with a satisfying substitution if the formula is satisfiable.

For instance, sat(or(P, not(P))) should succeed with P=true and, upon backtracking, P=false (in any order); sat(and(P, true)) should succeed with P=true.

* + fails, if the given formula is unsatisfiable. For instance, sat(and(P, not(P))) should fail.

1. **tautology:** Consider propositional boolean formulae from the previous question. Write a predicate tautology(F), where F is a term representing a boolean formula, that succeeds if and only if F is a tautology, i.e., true for every truth assignment to the propositional variables. For full credit, your predicate must not bind the variables in F.

Your program should be in a **single** file (see submission instructions below).

**Grading:**

The first three problems are worth 3 points each. Q4 is worth 4 points; Q5 and Q6 are worth 5 points each; finally Q7 is worth 2 points, for a total of 25 points.

**Submission:**

Your submission will consist of a **single** Prolog program file (name of the file and the file extension does not matter for the submission).

Submit your homework by filling out HW1 submission form on Blackboard (assignments area) before the deadline.