1. Write recursive rules to manipulate the following lists:
   a. **descSort** to sort the items of a list in descending order.
      e.g. descSort([2,1,3], FinalList) binds FinalList with [3,2,1], while descSort([1,4,3,2,3], ResList) binds ResList with [4,3,3,2,1]. [5]
   b. **occur** to find out the occurrence of a specific item within a list.
      e.g. occur(3,[1,2,3,4,3], Num) binds Num to 2, while occur(b,[1,2,3], Occ) binds Occ to 0. [5]
   c. **countDel** to count and delete all occurrences of a specific item in a list.
      e.g. countDel(2,[1,2,3,2,2],L,N) would bind L to [1,3] and N to 3, while countDel(4,[1,2,3],F,DL) would bind F to [1,2,3] and DL to 0. [5]
   d. **collCount** to collect same items and count number of occurrences.
      e.g. collCount([a,b,a,b,c,b],F) binds F to [(a,2),(b,3),(c,1)], while collCount([q,p,q,r],R) binds R to [(q,2),(p,1)(r,1)]. [5]
   e. **remDup** to list the items in descending order and remove duplicates.
      e.g. remDup([1,2,1,2,3,2],Ans) binds Ans to [3,2,1], while remDup([1,2,3,2,3,1,1],Final) binds Final to [3,2,1]. [5]

2. Write Prolog clauses to:
   a. **textify(ListToReplace,NewChar,InputList,OutputList)** that replaces all the occurrences of ListToReplace in InputList with the character NewChar to produce the OutputList. Assume that ListToReplace always has exactly three characters.
      e.g. textify([a,t,e], 8, [s,e,e,'',y,o,u,'',l,a,t,e,r,'',k,a,t,e], Ans) should bind Ans to [s,e,e,'',y,o,u,'',l,8,r,'',k,8]. [9]
   b. **wordTextify(WordToReplace,NewChar,InputList,OutputList)** that replaces all the occurrences of WordToReplace in InputList with the character NewChar to produce the OutputList.
      e.g. wordTextify(see, c, [see,you,later,kate],Ans) binds Ans to [c,you,later, kate]. [7]
   c. **fullText(WordListToReplace,NewChars,InputList,OutputList)** that replaces all the occurrences of WordListToReplace in InputList with the respective characters in the NewChars list to produce the OutputList. Assume that words within WordListToReplace always have exactly three characters.
      e.g. fullText([s,e,e,y,o,u,a,t,e],[c,u,8],[s,e,e,'',y,o,u,'',l,a,t,e,r,'',k,a,t,e], Ans) binds Ans to [c,'',u,'',l,8,r,'',k,8].
      [9]

3. Consider the following DCG:

Rules	Lexicon	
s --> np, vp.	det --> [the].	verb --> [brought].
np --> det, noun.	det --> [a].	prep --> [to].
np --> np, pp.	noun --> [waiter].	prep --> [of].
vp --> verb, np.	noun --> [meal].	
vp --> verb, np, pp.	noun --> [table].	
pp --> prep, np.	noun --> [day].	

a.
   i. Implement the above DCG in Prolog using Difference Lists; [5]
   ii. Write a query to check all possible sentences; [2]
   iii. Is the DCG syntactically correct? [2]

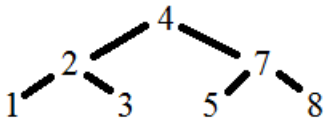
b. Write queries to check whether the following sentences are part of this grammar, and state whether they are or not.
   i. "the waiter brought the meal to the table";
   ii. "a meal brought by the waiter";
   iii. "the waiter brought the meal of the day". [9]

c. Draw the parse tree for one of the valid sentences from b. [7]

---

4. A binary tree can be defined in terms of 2 predicates, namely, an **emptyTree** that represent an empty binary tree or nil, and a **bTree(LeftTree, Node, RightTree)** that represents a binary tree with Left representing the left binary tree, Node is the root of the binary tree, and Right is the right binary tree. All the items in Left are less than or equal to the Node, and all the items in Right are greater than the Node.

e.g. bTree(bTree(bTree(nil,1, nil) ,2, bTree(nil,3, nil)),
4,
bTree(bTree(nil,5, nil) ,7, bTree(nil,8, nil)))



Write a Prolog code to implement the following predicates:

a. **insert(Item, OrigTree, FinalTree)** is true if the FinalTree is the binary tree resulting from Item being inserted into the binary tree OrigTree.
e.g. insert(5, bTree(bTree(nil,2, nil) ,4, bTree(nil,7, nil)), NewList) binds NewList to bTree(bTree(nil,2, nil) ,4, bTree(bTree(nil, 5,nil),7, nil)). [8]

b. **preorder(Tree, ValueList)** is true if ValueList is a list of nodes generated by a preorder traversal of the binary tree Tree.
e.g. preorder(bTree(bTree(nil,2, nil) ,4, bTree(nil,7, nil)), ListOfValues) binds ListOfValues to [2,4,7]. [8]

c. **search(Tree, Item)** is true if the Item is contained in the binary tree Tree.

e.g. search(3,bTree(bTree(bTree(nil,1,nil),2,bTree(nil,3,nil)),4,bTree(bTree (nil,5,nil),7,bTree(nil,8,nil))) returns Yes. [9]