; Program #9

; Austin Spencer

; manipulating a Bag of Strings

; All helper functions

;Helper function that will create a new pair to the bag

(define (newItem Item)

(cons Item 1)

)

;Helper function to add one to the integer value

(define (addInt Pair)

(cons (car Pair)(+ (cdr Pair) 1))

)

;Helper function to subtract one from the integer value

(define (subInt Pair)

(cons (car Pair)(- (cdr Pair) 1))

)

;Helper function to get the string of the pair

(define (getString Pair)

(car Pair)

)

;Helper function to get the integer value of the pair

(define (getInt Pair Pair2)

(cdr Pair)

)

;Helper function to add specific amount to value of the pair

(define (combine Pair)

(cons (car Pair)(+ (cdr Pair) Pair2))

)

;Helper function to compare two values and return the smallest

(define (compare Pair Pair2)

(if (< Pair Pair2)

(Pair)

(cons(car Pair)(Pair2))

;1) Return the number of times the given item is in the given list

(define (getBagCount List Item)

(if (null? List)

0

(if (string=? (car(car List) Item)

(cdr (car List))

(getBagCount (cdr List)) Item)

)

)

)

;2) Return a new bag that represents the result of inserting the given item in the given list

(define (insertBag List Item)

(if (null? List)

(newItem Item)

(if (string=? (car (car List)) Item)

(cons (addInt (car List))(cdr List))

(cons (car List)(insertBag (cdr List) Item))

)

)

)

;3) Return a new bag that represents the result of deleting one occurrence of the given item from the given list

(define (deleteBag List Item)

(if (null? List)

()

(if (string=? (getString (car List)) Item)

(if (> (getInt (car List)) 1)

(cons (subInt (car List)) (cdr List))

(cdr List)

)

(cons (car List) (deleteBag (cdr List) Item))

)

)

)

;4) Return a new bag that represents the result of deleting ALL occurrences of the given item from the given list

(define (deleteAllBag List Item)

(if (null? List)

()

(if (string=? (car (car List)) Item)

(cdr List)

(cons (car List) (deleteAllBag (cdr List) Item))

)

)

)

;5) Return a new bag that represents the result of the union of the two given bags (combine the contents of both - take the sum of the frequencies of each element in the two bags)

(define (unionBag ListA ListB)

(cond

((null? ListB)ListA)

((string=? (car ListA)(car ListB))(unionBag (cons (car ListA)(combine (cdr ListA)(cdr ListB)) (cdr ListB)

((#t) (if

(null? ListA)(unionBag (insertBag ListA (car ListB) (cdr ListB)

(#t) (cons(car ListA)(unionBag (cdr ListA) ListB)

;6) Return a new bag that represents the result of the intersection of the two given bags(take minimum frequency of each element in the two bags)

(define (intersectBag ListA ListB)

(define (intersectBag ListA ListB)

(cond

((null? ListB)ListA)

((string=? (car ListA)(car ListB))(intersectBag (cons (car ListA)(compare (cdr ListA)(cdr ListB)) (cdr ListB)

((#t) (if

(null? ListA)(intersectBag (insertBag ListA (car ListB) (cdr ListB)

(cons(car ListA)(intersectBag (cdr ListA) ListB)