**Assignment Goals**

* To make sure you can write data definitions for binary search trees.
* To make sure you can write programs over binary search trees.

**Preliminaries**

An important variant of the binary tree is the *binary search tree*. In a binary search tree, the tree is organized such that the key in any given node of the tree meets the conditions that the values of all keys in the node's left subtree are less than the key in the given node, and the values of all the keys in the node's right subtree are greater than the key in the given node. This organization makes the task of searching the tree more efficient (in terms of the number of comparisons needed to find a given key) than would be the case for a regular binary tree. This property (smaller values in the left subtree, greater values in the right subtree) is called the *invariant* of the binary search tree. You may copy the code from BTNode, Binary\_Tree, Binary\_Search\_Tree that are mentioned in chapter 8 of your text book.

*Make sure you name your class and functions exactly the same as the names given in the problems below. Otherwise, Web-CAT won't work and you'll lose points on the assignment.*

**The Assignment**

A credit card company keeps information on its cardholders. In order to provide efficient access to the data, the designer of the database decides to store information about cardholders in a binary search tree. In addition to the components that provide access to the left and right branches of the tree, each node in the tree contains a unique credit card number (**cc\_number** and the key value), customer**'s name, address,** and **email** address (all strings). A charge consists of thenameof the **business** (string) where the credit card was used, and the **amount** (double) that was charged. (The variable names are highlighted with bold face font)

1. Write the data definitions for **Contact**, and **Charge** and **ListOfCharge**. The name of the class for Contact should be **Contact**, and the name of the class for Charge should be **Charge**. Provide at least ten examples of Contact. Provide at least ten examples of ListOfCharge. When creating your examples, the address portion of your contact information doesn't have to be a complete address; for the purposes of this assignment, you may simply put in a street address (like "100 Institute Ave") instead of an entire address (like "100 Institute Ave, Worcester, MA 01609").
2. Create a class for **Contact**, **Charge, Customer class**.
3. Create a class **Customers** class which inherit from Binary\_Search\_Tree. Implement additional functions from the following assignment. (Notes: this is Customer**s** class)

;; a BST is one of

;; false

;; CustNode

;; a CustNode is a (make-customer Natural Contact Number ListOfCharge BST BST)

(define-struct customer (card-number contact limit charges left right))

1. Provide an example of a binary search tree containing information for at least 10 customers. Make sure you construct your example so that the customers are placed in the tree according to card number, satisfying the binary search tree invariant. When creating your examples, you don't have to use 16-digit credit card numbers like you might find on a real credit card; you may use small numbers like 123 or 9341. Don’t forget to pad “0” in front of the shorter credit card numbers. **Make sure you implement appropriate copy constructor for the classes**.
2. Write a function count-big-limits which consumes a binary search tree and a credit limit amount, and counts the number of customers in the tree who have a credit limit higher than the given amount.
3. Write a function any-over-limit? that consumes a binary search tree and produces true if any customer in the tree has an outstanding balance greater than their credit limit. A customer's balance is calculated by summing the amounts of all purchases that they've charged.
4. Write a function increase-limit. The function consumes a binary search tree, a credit card number, and an amount of money. The function returns a tree the same as the original, except that the credit limit of the customer with the given credit card number has been increased by the given amount. Your function should be written efficiently, taking advantage of the binary search tree invariant to minimize the number of comparisons needed to find the customer with the correct credit card number. You may assume that the given credit card number exists in the tree. Overload the **find** function in Binary\_Search\_Tree that take a credit card number and return the corresponding customer in the binary tree. You may modify the tree directly without returning the tree in your function.

This assignment is based on the following WPI assignment (homework 4, a18, 2018).

**Problem 1-4:**

(define-struct contact (name address email))

;; a Contact is a (make-contact String String String) where

;; name is the customer's name

;; address is the customer's physical address

;; email is the customer's email address

;; Contact examples

(define CONTACT1 (make-contact "John Smith" "1 Fake Way" "johnsmith@gmail.com"))

(define CONTACT2 (make-contact "Jane Doe" "2 Fake Lane" "janedoe@gmail.com"))

(define-struct charge (business amount))

;; a Charge is a (make-charge String Number) where

;; business is the name of the business

;; amount is the amount charged (as a Number)

;; a ListOfCharge is one of

;; empty

;; (cons Charge ListOfCharge)

;; ListOfCharge examples

(define LOC1 (cons (make-charge "Price Chopper" 5.99) (cons (make-charge "WPI" 100) empty)))

(define LOC2 (cons (make-charge "Papa Gino's" 9.99) empty))

;; a BST is one of

;; false

;; CustNode

;; a CustNode is a (make-customer Natural Contact Number ListOfCharge BST BST)

(define-struct customer (card-number contact limit charges left right))

;; a CustNode represents a customer in the database of customers where

;; card-number is the credit card number

;; contact represents the customer's contact information as a Contact

;; limit is the credit limit

;; charges is a list of outstanding charges as a ListOfCharge

;; left represents nodes to the left, and right represents nodes to the right

;; invariant: for any CustNode in the tree, the key value (card-number) is greater than the

;; key value of nodes to the left in CustNode's subtree, and is less than the key value

;; of any nodes on the right subtree

;; example of binary search tree

(define BST1 (make-customer 333 CONTACT1 100 LOC1

(make-customer 123 CONTACT2 150 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 105 LOC1

(make-customer 399 CONTACT1 125 LOC2 false false)

false)))

**;; problem 5**

;; count-big-limits: BST Number -> Natural

;; consumes a binary search tree and a credit limit amount, produces the number of customers

;; who have a credit limit higher than the given amount

(check-expect (count-big-limits false 100) 0) ;; empty tree

(check-expect (count-big-limits BST1 115) 3)

(check-expect (count-big-limits BST1 200) 0)

(check-expect (count-big-limits BST1 50) 5)

;; check-limit: Number Number -> Natural

;; takes in a customer's credit limit and a given credit limit, returns 1 if the customers's

;; credit limit is higher than the given limit, 0 otherwise.

(check-expect (check-limit 100 115) 0) ;; customer less than given

(check-expect (check-limit 100 100) 0) ;; customer same as given

(check-expect (check-limit 115 100) 1) ;; customer greater than given

**;; problem 6**

;; any-over-limit?: BST -> Boolean

;; consumes a binary search tree and produces true if any customer in that tree has exceeded

;; their credit limit with their total purchases (charges)

(check-expect (any-over-limit? false) false) ;;empty tree

(check-expect (any-over-limit? BST1) true) ;; some customers over limit

(check-expect (any-over-limit? (make-customer 333 CONTACT1 105.99 LOC1

(make-customer 123 CONTACT2 150 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 110 LOC1

(make-customer 399 CONTACT1 125 LOC2 false false)

false))) false) ;;all customers at or below limit, not over

(check-expect (any-over-limit? (make-customer 333 CONTACT1 110 LOC1

(make-customer 123 CONTACT2 150 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 110 LOC1

(make-customer 399 CONTACT1 125 LOC2 false false)

false))) false) ;;no customers at or over limit

;; sum-charges: ListOfCharge -> Number

;; consumes a ListOfCharge, produces the sum cost of those charges as a Number

(check-expect (sum-charges empty) 0) ;;empty list

(check-expect (sum-charges LOC2) 9.99) ;;one charge

(check-expect (sum-charges LOC1) 105.99) ;;multiple charges

**;; problem 7**

;; increase-limit: BST Natural Number -> BST

;; consumes a BST, a credit card number, and an amount of money. Function finds that

;; card number in the BST and changes that customer's credit limit, adding the amount of money

;; given to the limit. Function returns BST identical except for limit change

(check-expect (increase-limit BST1 333 50)

(make-customer 333 CONTACT1 150 LOC1

(make-customer 123 CONTACT2 150 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 105 LOC1

(make-customer 399 CONTACT1 125 LOC2 false false)

false)))

(check-expect (increase-limit BST1 399 25)

(make-customer 333 CONTACT1 100 LOC1

(make-customer 123 CONTACT2 150 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 105 LOC1

(make-customer 399 CONTACT1 150 LOC2 false false)

false)))

(check-expect (increase-limit BST1 123 10)

(make-customer 333 CONTACT1 100 LOC1

(make-customer 123 CONTACT2 160 LOC2

false

(make-customer 205 CONTACT1 125 LOC2 false false))

(make-customer 400 CONTACT2 105 LOC1

(make-customer 399 CONTACT1 125 LOC2 false false)

false)))

(check-expect (increase-limit false 0 0) false)

;; check-card-number: Customer Natural -> Boolean

;; consumes a customer and a card number, if the card number matches that of the customer,

;; return true, if not, return false

(check-expect (check-card-number BST1 333) true) ;; checks where it matches

(check-expect (check-card-number false 0) false) ;; checks a false tree

(check-expect (check-card-number BST1 399) false) ;;checks where it doesn't match