UNIVERSITY AT ALBANY
State University of New York

COLLEGE OF ENGINEERING AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

ICSI311 Principles of Programming Languages

Assignment 04 Created by Qi Wang

Table of Contents

Part I: General Information

- All assignments are individual assignments unless it is notified otherwise.
- All assignments must be submitted via Blackboard. No late submissions or e-mail submissions or hard copies will be accepted.
- Unlimited submission attempts will be allowed on Blackboard. **Only the last attempt will be graded.**
- Work will be rejected with no credit if
  - The work is late.
  - The work is not submitted properly (Blurry, wrong files, crashed files, files that can’t open, etc.).
  - The work is a copy or partial copy of others' work (such as work from another person or the Internet).
- Students must turn in their original work. Any cheating violation will be reported to the college. Students can help others by sharing ideas and should not allow others to copy their work.
- Documents to be submitted:
  - **Scheme source file(s) with inline comments** with file extension .rkt
  - **Test case document** with file extension .PDF
- Students are required to submit a design, all the error-free source files with Javadoc style inline comments and supporting files. Lack of any of the required items or programs with errors will result in a low credit or no credit.
- **Grades and feedback**: TAs will grade. Feedback and grades for properly submitted work will be posted on Blackboard. For questions regarding the feedback or the grade, students should reach out to their TAs first. Students have limited time/days from when a grade is posted to dispute the grade. Check email daily for the grade review notifications sent from the TAs. **Any grade dispute request after the dispute period will not be considered.**

Part II: Grading Rubric
See the requirements in part III.

Part III: Description

Goals:

- Review and develop a deep and comprehensive understanding of functional programming
- Review and develop a deep and comprehensive understanding of the divide-and-conquer technique and programming technique such as recursion
- Review and develop a deep and comprehensive understanding of data structures such as binary search trees

Instructions:

Please read the following requirements carefully. A program that doesn't meet the following requirements may be returned with 0 points awarded.

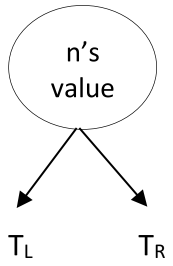
- Use Scheme as a **pure** functional programming language.
- **Only the following basic functions are allowed to be used for this assignment.**
  - Primitive Numeric Functions: `+`, `–`, `*`, `/`, `QUOTIENT`, and `MODULO`.
  - Defining Functions: `LAMBDA`, and `DEFINE`.
  - Numeric Predicate Functions: `=`, `<>`, `>`, `<`, `>=`, `<=`, `EVEN?`, `ODD?`, and `ZERO?`.
  - Logical Operator Functions: `AND`, `OR`, and `NOT`
  - Control Flow Functions: `IF`, `ELSE`, and `COND`.
  - List Functions: `QUOTE(')`, `CAR`, `CDR`, `CONS`, `LIST`, and `APPEND`.
  - Predicate Functions for Symbolic Atoms and Lists: `EQ?`, `EQV?`, `NULL?`, and `LIST?`.
  - LET functions: `LET`
- Use recursion, and not iteration.
- Adequate comments must be provided for the entire program.
- Adequate comments must be provided for each function in the program.
- Adequate comments must be provided for each logical block in a function for all functions in the program.
- Codes must be formatted properly using indentations to enhance readability.
- The program must be tested completely.
  - First, test each function completely.
    - Choose two various test cases at least, explain **how** to arrive at the output.
    - List the test cases and their results in comment format at the end of the program (for grading).
    - Write the explanations in a separate document using the template provided. You must name this document using the following naming convention: *FirstNameLastNameBSTTestCases* and submit a PDF for this document. A Word template is provided.
  - Next, test the entire program by invoking the functions in logical order.
    - Choose two various test cases at least, explain how to arrive the output.
    - List the test cases and their results in comment format at the end of the program (for grading).
    - Add the explanations to the above-mentioned document.
  - As mentioned in the previous steps, you must include the test cases and their results at the bottom of the program in comment format and complete the explanations in the test case document.
    - A program without test cases included at the end of program or without test case explanation document will be returned with 0 points awarded.
    - A submission of test case explanation document without the source codes will be returned with 0 points awarded.
- Submit the program with file extension .rkt and the test case document as a PDF.

Write a program that implements a binary search tree (BST). Assume that a BST organizes integers and can't contain duplicate values.

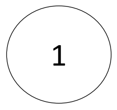
A binary search tree organizes its data by value. Each node *n* in a binary search tree satisfies the following properties:

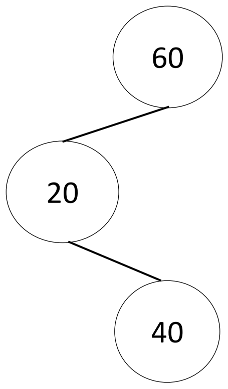
- n's value is greater than all values in its left subtree $T_{L.}$
- n's value is less than all values in its right subtree $T_{R.}$
- Both T_L and T_R are binary search trees.

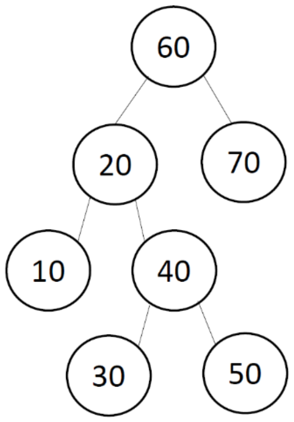
A non-empty binary search tree can be represented as a list of three elements - (a value, $T_{L,}$ T_R). The first element is a node's value(root), the second element is the node's left subtree, and the third element is a node's right subtree.



Non-empty left and right subtrees ($T_{L,}$ T_R) are binary search trees that are lists of three elements. An empty tree is represented as an empty list. The following examples show the trees and their list representations. Notice that the list representations are shown as pseudo codes and may not be in correct Scheme syntax.







`(1 () ())` `(60 (20 () (40 () ()) ) ())` What is the list representation?

The following shows the specifications of BST briefly. When implementing a function, tree status and/or input validation must be checked as needed.

Construct BST:

- Return an empty BST.
- Given an element, return a new BST contain a node containing the element.
- Given an element, a left subtree, and a right subtree, return a new BST if the three elements are valid, false otherwise.

Access BST:

- Given a BST, return the root of the tree.
- Given a BST, return the left subtree of the tree.

- Given a BST, return the right subtree of the tree.

Check for Emptiness:

- Given a BST, return true if the tree is empty, false otherwise.

Search:

- Given an element and a BST, return true if the element is in the tree, false otherwise.

Insertion:

- Given an element and a BST, if the element is already in the tree, return the tree. If the element isn't already in the tree, return a new binary search tree with the element appearing in its proper location. The original tree can't be changed.

BST Traversals:

- Given a BST, return a list containing all values in the tree in the order they are visited in a preorder traversal.
- Given a BST, return a list containing all values in the tree in the order they are visited in an inorder traversal.
- Given a BST, return a list containing all values in the tree in the order they are visited in a postorder traversal.

Test:

Test the design of the entire BST.