Project 3 – Functional Programming

Comp3071 Programming Languages

# Objective

**Be able to solve problems using the Lisp functional programming language.**

# Topics

**Pure functional programming**

* Recursion *(no iteration)*
* Function calls, parameters & return values *(No variables or assignment statements)*

# Instructions

1. **READ CAREFULLY AND FOLLOW ALL INSTRUCTIONS!!!!!!!!!!!!**
2. Download and install a **Lisp interpreter** if you have not done so already
3. **Write the functions** described in the table below using LISP and the pure-functional programming paradigm.
   * + Note that you may need/use them as building blocks in the project that you select in D.
     + Double check that the parens match in my examples before testing (possible I may have missed some).

|  |  |
| --- | --- |
| Function Description | Examples |
| Write a recursive function add-numbers, which is passed a mixed list of symbols, lists and numbers, and returns the sum of all numbers at all levels. | **(add-numbers** '(3 60 (Anthony (Adam 5)) Oresti 2 Yong)) 🡪 70  (**add-numbers** '(Jared 10 5 Tologon)) 🡪 15 |
| Write a recursive function count-all, to count all the symbols in a list, at all levels. | (**count-all**  '(Emma (Nicholas YanWen) Doris Harrison)) 🡪 5  (**count-all** '((Ely William Crissy) David (Chris (Stephen Khadija)) Jamie Tom) ) 🡪 9 |
| Write a recursive function remove-all, which takes 2 arguments, an atom and a list, and returns the list with all instances of the atom removed. | (**remove-all** ‘Andy ‘(Austen Andy Khalil (Andy John) Anthony)) 🡪 (Austen Khalil (John) Anthony)  (**remove-all** ‘u ‘(u e u e u e u)) 🡪 (e e e) |

1. **SELECT ONE of the following 3 project options and implement your solution using Lisp and the pure functional programming paradigm.**
2. **SIMPLIFY arithmetic expressions *(don’t do the actual calculations):***

|  |  |
| --- | --- |
| Background | Suppose that an arithmetic expression is any functional form using only + and \* as functions and using only constant numbers, variables (symbols), and (nested) arithmetic expressions as arguments. |
| Instructions | Write function **simplify** that takes **any** arithmetic expression as described above *(not just the examples shown)* and returns a new function in which the following improvements are made, if they are possible:   1. **Multiplication sub-expression** 2. With a 0 as an argument, sub-expression is replaced by 0: (\* 3 5 0) 🡪 0 3. With a 1 as an argument, the 1 is removed: (\* 1 2 6) 🡪 (\* 2 6). If only one argument remains then the sub-expression is replaced by that argument: (\* 1 6) 🡪 6 4. **Addition sub-expression** 5. Any occurrence of 0 is eliminated: (+ 0 2 7 7) 🡪 (+ 2 7 7). If only one argument remains, the sub-expression is eliminated: (+ 0 7) 🡪 7 |
| Examples | **(simplify ‘(+ 1 2 0 ) 🡪 (+ 1 2)**  Note: These are only examples; your function should work on ANY arithmetic expression in the form described above (that I might test it on).  **(simplify ‘(\* 1 2 ) 🡪 2**  (**simplify** **‘(\* (+ 6 0) (\* 1 6 2)))** 🡪 (\* 6 (\* 6 2)) |

|  |  |
| --- | --- |
| Background | Binary Search Trees have nodes with up to 3 elements: first is the value, second is the left sub-tree (which can be empty) and third is the right sub-tree (which can be empty). Value in node is greater than values in left sub-tree and less than or equal to values in right sub-tree. |
| Instructions | Write a lisp function (**buildBST list)**  **Parameter**   * List of unordered elements   **Behavior**   * Returns a list representing a Binary Search Tree.   Write a lisp function (**displayBST** list)  **Parameters**   * List representing a Binary Search Tree   **Behavior**   * Traverses and prints the node values in order (Depth First Search, displaying nodes in order).   Write a lisp function (**addToBST e list)**  **Parameters**   * New element * List representing a Binary Search Tree   **Behavior:**   * Adds new element to the Binary Search Tree (must retain the property of a binary search tree) * Displays the new tree after the element is added. |
| Examples  Note: These are only examples; your functions should work on ANY input in form expected (that I might test it on). | **(buildBST** ‘(6 9 2 1 7)) 🡪 (6 (2 (1) ()) (9 (7) ()))  **(displayBST** ‘(6 (2 (1) ()) (9 (7) ())))🡪 (1 2 6 7 9)  (**addToBST** ‘4 ‘(6 (2 (1) ()) (9 (7) ()))) 🡪 (6 (2 (1) (4)) (9 (7) ())) |

1. **Binary Search Trees**

1. **Student Choice**

This option is for students who are self-directed and want more of a challenge. You may propose your own project of interest; if you need ideas or resources, please ask. If you have an interest in AI this will provide an opportunity for you to do a specific project of interest.

1. **Demo your working functions to the instructor (not to the whole class) in lab.**

# rubric

|  |  |
| --- | --- |
| *Requirement* | *Percent* |
| *Add-numbers* | *10* |
| *Count-all* | *10* |
| *Remove-all* | *10* |
| *Select one: Simplify OR BST OR student choice* | *70* |

# Submitting your work

1. Make sure that your name is in ***all***of your project files.
2. If you have **more than one file** for your solution, make a **.zip file** for your project
3. In **Blackboard**, attach your solution file to the submission for this assignment.