CS 3210 – Principles of Programming Languages

Project 3: Functional programming and the Common Lisp development environment

The purpose of the project is to familiarize you with functional programming and the Common Lisp development environment. Please finish the functions below and submit your lisp source code file on Canvas.

Note: Use only the following standard Lisp functions, macros, operators, and constants in your definitions, along with any previously completed functions in this project:

- T
- NIL
- IF
- WHEN
- COND
- NOT
- AND
- OR
- EQUAL
- CONS
- LIST
- CAR
- CDR
- FIRST
- SECOND
- THIRD
- LENGTH
- DEFUN
- LABELS
- LET
- LET*
- FUNCALL
- QUOTE
- any arithmetic operator or relation (+, -, *, /, <, <=, >, >=, =)
- any numerical constant

```
;; Return T if item is a member of set.
;; Return NIL if item is not a member of set.
;; The type of set is list.
;; Examples:
;;  (set-member '(1 2) 1) => T
;;  (set-member '(1 2) 3) =>  NIL
(defun set-member (set item)
  ;;Your implementation go here
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the union of set-1 and set-2.
;; The result should contain no duplicates.
;; Assume set-1 contains no duplicates and set-2 contains no duplicates.
;; Examples:
;;   (set-union '(1 2) '(2 4)) => '(1 2 4)
(defun set-union (set-1 set-2)
  ;;Your implementation go here
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the intersection of set-1 and set-2.
;; The result should contain no duplicates.
;; Assume set-1 contains no duplicates and set-2 contains no duplicates.
;; Examples:
;;   (set-intersection '(1 2) '(2 4)) => '(2)
(defun set-intersection (set-1 set-2)
  ;;Your implementation go here
)
```

```lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the difference of set-1 and set-2.
;; The result should contain no duplicates.
;; Assume set-1 contains no duplicates and set-2 contains no duplicates.
;;
;; Examples:
;;   (set-diff '(1 2) '(2 4)) => '(1)
(defun set-diff (set-1 set-2)
  ;;Your implementation go here
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the exclusive or of a and b
;;
;; Examples:
;;  (boolean-xor t nil) => t
;;  (boolean-xor nil nil) => nil
(defun boolean-xor (a b)
  ;;Your implementation go here
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the implication of a and b
;;
;; Examples:
;;  (boolean-implies t nil) => nil
;;  (boolean-implies nil nil) => t
(defun boolean-implies (a b)
;;<Your implementation go here >
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Return the bi-implication (if and only if) of a and b
;;
;; Examples:
;;  (boolean-iff t nil) => nil
;;  (boolean-iff nil nil) => t
(defun boolean-iff (a b)
;;<Your implementation go here >
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Evaluate a boolean expression.
;; Handle NOT, AND, OR, XOR, IMPLIES, and IFF.
;;
;; Examples:
;;  (boolean-eval '(and t nil)) => nil
;;  (boolean-eval '(and t (or nil t)) => t
(defun boolean-eval (exp)
;;<Your implementation go here >
)
```