Selected String and Character Functions

- `(char=? c d)` produces `true` if characters `c` and `d` are equal, `false` otherwise
- `(string->list s)` produces a list of the characters in string `s`
- `(list->string lst)` produces a string from the characters in list `lst`
- `(string=? s t)` produces `true` if the strings `s` and `t` are equal, `false` otherwise
- `(string-length s)` produces the number of characters in string `s`
- `(string-append s1 s2 ... sn)` produces string formed by concatenating `s1` to `s2` to `s3` to ... `sn`
- `(substring s p1 p2)` produces the string containing all the characters from position `p1` to `p2-1`, inclusive, where the first character is at position 0
- `(substring s p1)` produces the string containing all characters from position `p1` to end of `s`
- `(string? s)` produces `true` if `s` is a string, `false` otherwise
- `(char? c)` produces `true` if `c` is a character, `false` otherwise
- `(string-ref s n)` produces the character at position `n` in the string `s`, where the first character is at position 0
- `(number->string n)` produces the number `n` as a string

Abstract List Functions

- `(build-list n f)` produces a list of length `n`, obtained by applying `f` to 0, 1, 2, ..., `n-1`
- `(filter pred lst)` produces a list of the elements `x`, in `lst`, for which `(pred x) => true`
- `(foldr combine base lst)` produces the value of `(combine x1 (combine x2 (... (combine xN base) ...)))` where `lst` contains `x1,x2,...,xN`
- `(map f lst)` produces a list with the same length as `lst` obtained by applying `f` to each element in `lst`
- `(andmap pred lst)` produces `true` if `pred` produces `true` when applied to all elements in `lst`, and `false` otherwise
- `(ormap pred lst)` produces `true` if `pred` produces `true` when applied to at least one element in `lst`, and `false` otherwise
- `(sort lst comp)` produces sorted version of `lst`, using `comp` as comparison function

Other Selected Functions

- `(add1 n)` produces `n+1`
- `(sub1 n)` produces `n-1`
- `(append l1 l2 ... ln)` consumes two or more lists `l1,l2,...,ln`, and produces a list of all the elements in `l1,l2,...,ln`, in that order
- `(check-expect act exp)` test passes if `act` and `exp` are equal, fails otherwise
- `(equal? e1 e2)` produces `true` if `e1` and `e2` are equal, `false` otherwise
- `(symbol=? s t)` produces `true` if symbols `s` and `t` are equal, `false` otherwise
- `(empty? lst)` produces `true` if `lst` is the empty list, `false` otherwise
- `(member? x lst)` produces `true` if `x` is in `lst`, `false` otherwise
- `(length lst)` produces the number of elements in the list `lst`
- `(max x y)` produces `x` if `x >= y`, and `y` otherwise
- `(min x y)` produces `x` if `x <= y`, and `y` otherwise
- `(sqr x)` produces the value `x` squared
- `(sqrt x)` produces the square root of `x`
- `(number? x)` produces `true` if `x` is a number, `false` otherwise
Some functions and data definitions on this sheet may not appear in the exam solutions.

(integer? x) produces true if x is an integer, false otherwise
(negative? x) produces true if x < 0, false otherwise
(positive? x) produces true if x > 0, false otherwise
(zero? x) produces true if x is equal to 0, false otherwise
(even? x) produces true if x is an even number, false otherwise
(odd? x) produces true if x is an odd number, false otherwise
(boolean? x) produces true if x is a boolean value, false otherwise
(quotient n m) produces the quotient when n is divided by m
(remainder n m) produces the remainder when n is divided by m
(expt b p) produces the value of b to the exponent p
(range a b c) produces the list from a to b, but not including b, stepping by c, that is, the list contains a, a+c, a+2c, ...

Data Definitions

An Association (As) is a (list Num Str), where the first element in an As is the key.
An Association List (AL) is a (listof As).
Note that there are no duplicate keys in an AL.

(define-struct binode (op arg1 arg2))
A Binary Arithmetic Expression (BinExp) is one of
• Num
• (make-binode (anyof '+ '-' '*' '/') BinExp BinExp)

(define-struct node (key val left right))
A Binary Tree (BT) is one of
• empty
• (make-node Nat Str BT BT)

A Binary Search Tree (BST) is one of
• empty
• (make-node Nat Str BST BST)
And satisfies the ordering property recursively:
• Every key in left is less than key
• Every key in right is greater than key
Note that there are no duplicate keys in a BST.

An Atom is an (anyof Num Str Sym)
A Leaf-Labelled Tree (LLT) is one of
• empty
• (cons Atom LLT)
• (cons LLT LLT) where the first LLT is nonempty