COMP_SCI 111 - Quiz Glossary

Arithmetic

`(+ numbers …)`
numbers … → number
Returns the sum of the given numbers.

`(- number number)`
number number → number
Returns the difference of two numbers.

`(- number)`
number → number
Returns the number times -1.

`(* number number)`, `(/ number number)`
number number → number
Returns the specified product (or quotient) of the specified numbers.

`(quotient integer integer)`
number number → number
Returns the quotient of the two integers, rounded down to the nearest integer.

`(abs number)`
number → number
Returns the absolute value of number, i.e. the number with the sign erased.

`(sin number)`, `(cos number)`, `(sqrt number)`
number → number
Returns the sine, cosine, or square root of the number, respectively.

`(max numbers …)`, `(min numbers …)`
number … → number
Returns the maximum/minimum of the *numbers*.

Comparisons

`(string=? string1 string2)`
string string → Boolean
Returns true if *string1* and *string2* are equivalent.

`(= number1 number2)`
number number → Boolean
Returns true if numbers are equal.

`(< number number)`, `(> number number)`,
`(>= number number)`, `(<= number number)`
number number → Boolean
Returns true if *first number* is less than, greater than, greater than or equal to, or less than or equal to, *second number*, respectively.

Other predicates

`(and booleans …)`, `(or booleans …)`
Booleans … → Boolean
Returns true if all/any of the *booleans* are true.

`(not boolean)`
Boolean → Boolean
Returns true if input is false, or false if input is true.

`(odd? number)`, `(even? number)`
number → Boolean
Returns true if number is odd/even, else false.

`(number? object)`, `(integer? object)`,
`(string? object)`, `(list? object)`
any → Boolean
Returns true if *object* is of that type, otherwise false.

Images

`(color red green blue)`
number number number → color
Returns a color with specified red, green, and blue parts.

`(rectangle width height mode color)`
`(ellipse width height mode color)`
number number string color → image
Returns the shape with the specified *width* and *height* (numbers), *mode* (either "outline" or "solid") and *color*.

`(square size mode color)`
`(circle size mode color)`
number string color → image
Returns a square or circle of the specified *size* (numbers), *mode* (either "outline" or "solid") and *color*.

`(regular-polygon length sides mode color)`
number number string color → image
Returns a regular polygon of the specified *length*, number of *sides*, *mode* and *color*.

`(overlay images …)`, `(beside images …)`,
`(above images …)`
image … → image
Returns an image composed of all the input images.

`(iterated-overlay function count)`
`(iterated-beside function count)`
`(iterated-above function count)`
(number → image) number → image
Function should be a function that takes a number as input and returns an image. Uses function *count* times with arguments starting at 0 and going to *count*-1, returning the composite of all the images.

`(`**`scale`** *`magnification image …`*`)`
`(`**`rotate`** *`degree image …`*`)`
number image → image
Returns a composite picture of all the specified pictures and scales/rotates it by the specified amounts.

`empty-image`
image
A blank image.

Lists

`(`**`list`** *`elements …`*`)`
X … → (listof X)
Returns a list with all the specified *elements*, in order.

`(`**`append`** *`lists …`*`)`
(listof X) … → (listof X)
Returns one long list containing all the elements of all the *lists*, in order. Thus `(append (list 1 2) (list 3 4))` returns the list `(1 2 3 4)`.

`(`**`list-ref`** *`list position`*`)`
(listof X) number → X
Returns the element of *list* at the specified *position* (0 is first element, 1 is second, etc.).

`(`**`first`** *`list`*`), (`**`second`** *`list`*`), etc.`
(listof X) → X
Returns the first (or second, etc.) element of the *list*. If *list* is the empty list, it throws an exception.

`(`**`cons`** *`element list`*`)`
X (listof X) → (listof X)
Returns a new list starting with *element* followed by all the elements of *list*, in order. Thus `(cons 1 (list 0 0))` returns the list: `(list 1 0 0)`.

`(`**`rest`** *`list`*`)`
(listof X) → (listof X)
Returns a list containing all but the first element of *list*. Thus `(rest (list 1 2 3))` returns the list: `(list 2 3)`. If *list* is the empty list, it throws an exception.

`(`**`empty?`** *`list`*`)`
list → boolean
Returns true if *list* has no elements, else returns false.

`(`**`length`** *`list`*`)`
list → number
Returns the number of items in *list*.

`(`**`map`** *`function list`*`)`
(In → Out) (listof In) → (listof Out)
Calls function on each element of list and returns all the results as a list. In other words, `(map func (list 1 2 3))` behaves like `(list (func 1) (func 2) (func 3))`.

`(`**`for-each`** *`function list`*`)`
(In → Out) (listof In) → void
Calls function on each element of list, returns nothing.

`(`**`filter`** *`function list`*`)`
(X → boolean) (listof X) → (listof X)
Returns a new list consisting of only those elements of the original *list* for which *function* returns true. If *function* returns a value other than true or false, it will produce an exception.

`(`**`foldl`** *`function start list`*`)`
`(`**`foldr`** *`function start list`*`)`
(X Y → Y) Y (listof X) → Y
Applies *function* pairwise to all the elements of *list*. So folding + over a list of numbers starting at 0 will return the sum of all the numbers. If *list* is empty, fold will just return *start*. foldl processes the list elements left-to-right, and foldr processes them right-to-left.

`(`**`apply`** *`function list`*`)`
function list → any
Calls *function* with all the elements of *list* (in order) as arguments to the function. `(apply + (list 1 2 3))` behaves like `(+ 1 2 3)`.

`(`**`andmap`** *`pred list`*`),(`**`ormap`** *`predicate list`*`)`
(X → boolean) (listof X) → boolean
Calls *predicate* on eeach element of *list*. Ormap returns true if *predicate* returns true for at least one element of *list*. Andmap returns true if *predicate* returns true for every element of *list*. If *predicate* returning a value other than true or false, both to produce an exception.

`(`**`member`** *`item list`*`)`
X (listof X) → Boolean
True if and only if *item* is contained in *list* else false.

`(`**`remove-all`** *`item list`*`)`
X (listof X) → (listof X)
Returns the *list* with every occurrence of *item* removed.

Strings

`(`**`string-append`** *`strings …`*`)`
string … → string
Returns a new string containing all the text from *strings*.

`(`**`string-length`** *`string`*`)`
string → number
Returns the number of characters in the input.

`(`**`printf`** *`string things-to-print…`*`)`
string any … → void
Displays *things-to-print* according to *string* template.