**LISP Solutions**

**Prog 1: Write a program in Lisp to perform the functions of a Simple calculator.**

(defun cal()

(terpri)

(princ "enter 1st no. ")

(setq n1(read))

(princ "enter 2nd no. ")

(setq n2(read))

(princ "\n1.SUM")

(princ "\n2.DIFFERENCE")

(princ "\n3.PRODUCT")

(princ "\n4.QUOTIENT")

(princ "\n1.Enter your choice [1, 2, 3, 4]:")

(setq ch(read))

(cond ((and (eq n2 0) (eq ch 4)) (princ "\n Division by zero NOT ALLOWED") (terpri))

(t (calculate n1 n2 ch 0))

))

(defun calculate(n1 n2 ch result)

(cond ((eq ch 1)(+ n1 n2))

((eq ch 2)(- n1 n2))

((eq ch 3)(\* n1 n2))

((eq ch 4)(/ n1 n2))

(t (princ "\nWrong option given")(terpri))

))

**Prog2: Write a program in Lisp to find the average of n numbers (with and without using List).**

\*\*check

(defun avg()

(princ "enter N: ")

(setq n (read)) (setq num n)

(setq sum 0)

(loop

(setq temp (read))

(setq sum (+ sum temp))

(setq n (- n 1))

(if (eq n 0) (return))

)

(setq a (/ sum num))

(princ a))

**Prog3: Write a program in Lisp to find Factorial of a number.**

(defun fact()

(princ "enter the number: ")

(setq a(read))

(setq f 1)

(loop

(if (eq a 0)(return f))

(setq f (\* f a))

(setq a (- a 1))

)

)

**Prog4: Write a program in Lisp to find Factorial in a range.**

\*\*check

(defun range()

(princ "enter the limits: \n")

(setq l (read))(setq h (read))

(setq i l)

(loop

(if (eq i h)(return))

(princ i)(print "factorial=")

(setq l i)

(princ (fact l))

(terpri)

(setq i (+ 1 i))

))

(defun fact(l)

(setq f 1)

(loop

(if (eq l 0)(return f))

(setq f (\* f l))(setq l (- l 1))

))

**Prog5: Write a program in Lisp to find area of a circle of given radius.**

(defun ar()

(princ "Enter radius:")

(setq r (read))

(setq a (\* 3.14 (\* r r)))

(princ "\nArea=")(princ a)

)

**Prog6: Write a program in Lisp to get the sum of i) ‘n’ even/odd number ii) even/odd numbers up to ‘n’.**

1. **Sum of ‘n’ odd numbers**

(defun sum(m)

(cond ((< m 0) (princ "invalid input"))

(t (sum1 1 0 m 1))

)

)

(defun sum1(n s m c)

(cond ((> c m) s)

(t (sum1(+ n 2) (+ s n) m (+ 1 c)))

)

)

**Sum of ‘n’ even numbers**

(defun sum(m)

(cond ((< m 0) (princ "invalid input"))

(t (sum1 2 0 m 1))

)

)

(defun sum1(n s m c)

(cond ((> c m) s)

(t (sum1(+ n 2) (+ s n) m (+ 1 c)))

)

)

1. **Sum of odd numbers up to ‘n’**

(defun sum(m)

(cond ((< m 0) (princ "invalid input"))

(t (sum1 1 0 m))

)

)

(defun sum1(n s m)

(cond ((> n m) s)

(t (sum1(+ n 2) (+ s n) m))

)

)

**Sum of even numbers up to ‘n’**

(defun sum(m)

(cond ((< m 0) (princ "invalid input"))

(t (sum1 2 0 m))

)

)

(defun sum1(n s m)

(cond ((> n m) s)

(t (sum1(+ n 2) (+ s n) m))

)

)

**Prog7: Write a Lisp program to get the maximum element of a list.**

(defun maxm(y)

(setq s (car y))

(setq ls (cdr y))

(setq f 0)

(princ "max no\n")

(maxcal ls s )

)

(defun maxcal(l2 s2 )

(setq a2 (car l2))

( cond((null l2 )s2)

( t (cond(( < s2 a2)(setq s2 a2)(maxcal (cdr l2) s2 ))

( t (maxcal (cdr l2) s2 )))

)

)

)

Output:

(maxm ‘(8 9 5 7))

9

**Prog8: Write a Lisp program to get the minimum element of a list.**

(defun min(y)

(setq s (car y))

(setq ls (cdr y))

(mincal ls s )

)

(defun mincal(l1 s1 )

(setq a1 (car l1))

( cond((null l1 )s1)

(t (cond(( > s1 a1)(setq s1 a1)(mincal (cdr l1) s1 ))

(t (mincal (cdr l1) s1 )))

)

)

)

Output:

(min ‘(8 5 7))

5

**Prog9: Write a program in Lisp to find the Fibonacci Series up to nth term.**

-do- hint: use loop

**Prog10: Write a Lisp program to get the nth term of a list.**

(defun nth(n lst)

(cond ((null lst) nil)

((eq n 1) (car lst))

(t (nth (- n 1) (cdr lst)))

)

)

Output:

(nth 6 ‘(6 7 99 0 54 34 22 1 5 7 14 22))

34

**Prog11: Write a Lisp program to get the sum of ‘n’ natural numbers.**

(defun sumn(n)

(cond ((< n 0) (princ "invalid input"))

(t (sum1 n 0))

)

)

(defun sum1(n s)

(cond ((equal n 0) s)

(t (sum1(- n 1) (+ s n )))

)

)

**Prog12: Write a program in Lisp to let the union of two lists.**

(defun union (x y)

(cond ((null x ) y)

( t ( union1 x y x ))

)

)

(defun union1 ( x y z)

( cond (( null y ) z )

((find (car y) x) ( union1 x (cdr y ) z ))

( t ( union1 x (cdr y) (append z ( list (car y)))))

)

)

(defun find(x y)

(cond (( null y ) nil)

(( eq x ( car y )) t)

( t ( find x ( cdr y)))

)

)

Output:

(union ‘(3 5 6) ‘(1 2 4))

(3 5 6 1 2 4)

**Prog13: Write a program in Lisp to let the intersection of two lists.**

(defun intersection(x y)

(intersect x y '())

)

(defun intersect(x y z)

(cond ((null x) z)

((find (car x) y) (intersect (cdr x) y ( append z (list (car x)))))

(t (intersect (cdr x) y z))

)

)

(defun find(x y)

(cond ((null y) nil)

((eq x (car y)) t)

(t (find x (cdr y)))

)

)

Output:

(intersection ‘(2 3 5 6) ‘(1 2 3))

(2 3)

**Prog14: Write a program in Lisp to get the reverse of a list.**

(defun rev(L)

(cond ((null L) nil)

(t (insertend (car L) (rev (cdr L))))

)

)

(defun insertend (E L)

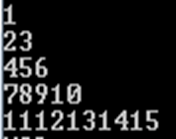
(cond ((null L) (list E))

(t (cons (car L) (insertend E (cdr L))))

)

)

**Prog15: Write programs in Lisp to generate the following patterns:**

****

(defun tri(n)

(setq k 1)

(dotimes (i n)

(dotimes (j (+ i 1))

(princ k)

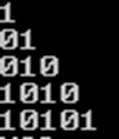
(setq k (+ k 1))

)

(princ "\n")

)

)



(defun tri(n)

(setq k 1)

(dotimes (i n)

(dotimes (j (+ i 1))

(cond ((eq (rem k 2) 0) (princ "0"))

(t (princ "1"))

)

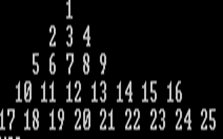
(setq k (+ k 1))

)

(princ "\n")

)

)



(defun tri(n)

(setq l 1)

(dotimes (i n)

(setq c (+ (\* 2 i) 1))

(setq g (- (- n i) 1))

(dotimes (j g)

(princ " ")

)

(dotimes (k c)

(princ l)(princ " ")

(setq l (+ l 1))

)

(princ "\n")

)

)



(defun tri(n)

(setq l 1)

(dotimes (i n)

(setq c (+ (\* 2 i) 1))

(setq g (- (- n i) 1))

(dotimes (j g)

(princ " ")

)

(dotimes (k c)

(princ (rem l 2))

(setq l (+ l 1))

)

(princ "\n")

)

)



(defun pat(n)

(setq i 1)

(dotimes (i n)

(setq j 1)

(dotimes (j (- n i))

(princ " ")

)

(setq k 1)

(dotimes (k i)

(princ "1")

)

(princ "\n")

)

)