עבודה 2 -חלקים תיאורטיים עקרונות שפות תכנות.

שאלה 1 :

1.1) צורות מיוחדות הכרחיות בשפות תכנות כדי להגיר מבנים (מבחינה סמנטית) עבור ביטויים מורכבים להם אין ברירות מחדל באינטרפרטר. את האופרטורים פרימיטיביים נחשב אותם ע"י חוקי האופרטור שמוגדרים באינטרפרטר-ע״פ העקרון שהערך של אופרטור פרימיטיבי זהה לסימון שלו בתחביר. לעומת זאת, צורות מיוחדות נבנות ונוצר ערך מביטוי על פי כללים מיוחדים ושונים מאופרטורים פרימיטביים. ההגדרה של כל צורה מיוחדת מוגדרת כמבנה בשפה.
לדוגמה – ((3 2 +)define x). זוהי צורה מיוחדת המחשבת את הביטוי (3 2 +) וע"י ביינדינג מוסיפה אותו לסביבה בצורה של זוג סדור <x,5>. זהו למעשה חוק הגזירה של הצורה המיוחדת define בשפה.

1.2) תוכנית ב-1L שניתן לבצע במקביל:
(2 1 +)
(4 3 +)

תוכנית ב1L שלא ניתן לבצע במקביל:
(define a 3)
(2 a *)

1.3) ב-1L קיימת צורה מיוחדת יחידה – define ולכן בשפה 0L אין אף צורה מיוחדת.
במידה והתוכנית ב1L לא מכילה define אז היא גם תקפה בשפה 0L.
במידה והתוכנית ב-1L כן מכילה define אז נוכל להמיר ע״י הצבה פשוטה כל משתנה עליו בוצע define בערך המתאים בהתאם לתוכנית ב-1L.

L1: (define x 2) ⟶ L0: (4 2 *)
(4 x *)

1.4) ב-2L קיימות הצורות המיוחדות הבאות : define, lambda, if. בעוד לשפה 20L יש את הצורות המיוחדות lambda and if בלבד.
כאשר נבנה תוכנית רקורסיבית ב-2L שמטרתה חישוב מורכב או יצירת לולאה שמצבעת ברקורסיה אנו משתמשים בצורה המיוחדת define,lambda ע״מ לבצע פעולות אלו. לא נוכל להמיר אותו לשפה 20L מכיוון שללא define לא נוכל לבצע את הקריאה הרקורסיבית הנדרשת.

1.5) **Map** – ניתן להריץ בצורה מקבילית. פונקציה זו מפעילה פונקציה מסוימת שמתקבלת בקלט על כל איבר במערך בצורה בלתי תלויה ולכן גם בהרצה על כל איבר בנפרד ושילוב התוצאות למערך אחד אין סכנה פעגישה בערכים המוחזרים.
Reduce – לא ניתן להריץ בצורה מקבילית. לפונקציה זו יש להזין ערך התחלתי אשר כל איברי המערך תלויים בו בנוסף ישנו איבר מיוחד בפונקציה והוא acc אשר שומר את התוצאה המתקבלת מביצוע הפעולות עד כה. ולכן קיימות פונקציות שאינן קומטטביות אשר חישובים עליהם בסדר שונה עלולות להחזיר תשובה שונה מהתוצאה הצפויה (לדוגמה – חילוק).
Filter – ניתן להריץ בצורה מקבילית. פונקציה זו מפעילה תנאי (predicate) על כל איבר במערך בצורה בלתי תלויה. לכן לא קיים סיכון לפגיעה בתוצאה הצפויה.
All – ניתן להריץ בצורה מקבילית . פונקציה זו מפעילה predicate אשר מחזיר אמת או שקר עבור כל איבר במערך בצורה בלתי תלויה (כאשר תוצאתו הסופית שקולה לוגית לפעולה AND) ולכן לא קיים סיכון לפגיעה בתוצאה הצפויה.
Compose – לא ניתן להריץ בצורה מקבילית. הרכבת/שרשור פונקציות אינה פעולה קומטטיבית ולכן ייתכנו פונקציות שהרצתן בצורה מקבילית עלולה להחזיר תוצאה שונה מתוצאה בריצה סדרתית.

1.6) יוחזר הערך 9.
כאשר נקרא ל ((define p34 (pair 3 4) אנו קוראים לבנאי של pair המוגדר ע״י השפה L31 . ניתן לראות בשורה השנייה של התוכנית כי c=2 ולכן בעת הקריאה לבנאי, הסביבה (ENV) תכיל את ההגדרה של c בתור 2.
סך הכל נקבל מצב בו a=3, b=4, c=2 ולכן יוחזר סכומם השווה ל-9. בעצם הספרה 5 אינה משפיעה על התוצאה במקרה זה.

שאלה 2 :

q.2.1

```
;;Signature: append(lst1 lst2)
;;Type: [ List(T2) * List(T1) -> List(T1|T2) ]
;; Purpose: append 2 lists
;; Pre-condition: none
;; Tests:(append '(1 2 ) '(3 4 ))-> '(1 2 3 4)
```

q.2.2

```
;;Signature: reverse(lst1)
;;Type: [List(T1) -> List(T1) ]
;; Purpose: return reverse list
;; Pre-condition: none
;; Tests:(reverse '(1 2 3)-> '(3 2 1))
```

q.2.3

```
;;Signature: duplicate-items(lst dup-count)
;;Type: [List(T1)*List(numbers) -> List(T1) ]
;; Purpose: get a list “lst” that duplicate the items according to the dup-count list.
;; Pre-condition: dup-count not equal to ’()
;; Tests:(reverse '(1 2 3) '(1 2 1) -> '(1 2 2 3))
```

q.2.4

```
;;Signature: payment(n coins-lst)
;;Type: [number*List(numbers) -> List(numbers) ]
```

;; Purpose: gets a sum of money and list of available coins, and returns the number of possible ways to pay the money with these coins

;; Pre-condition: n isn't negative.

;; Tests:(payment 10 '(5 5 10)) → 2)

<u>q.2.5</u>

;;Signature: compose-n(f n)

;;Type: [(T1=>T1) * number : T1)]

;; Purpose: compose n times f.

;; Pre-condition: f is unary function

;; Tests:(define mul8 (compose-n (lambda (x) (* 2 x)) 3))

;;(mul8 3)->24