**ממ״ן 14**

**שאלה 1:**

1. סכימת התרגום הנדרשת:
2. סכימת תרגום כנדרש (למען הפשטות: ערך ברירת מחדל של כל התכונות הינו 0) משתמשת בתכונות הבאות:

|  |  |
| --- | --- |
| כמה תווים בערך המשתנה A/B צריך להימנע מלהדפיס – תכונה מורשת. |  |
| כמה תווי ‘a’ ישנם בערך המשתנה A/B. תכונה נבנית. |  |
| מונה לכל תו a,b,c בהתאמה – כמה הופעות של התו מופיעות במחרוזת שנקראה משמאל – עד עכשיו. תכונה מורשת. |  |
| מונה לכל תו a,b,c בהתאמה – כמה הופעות של התו מופיעות במחרוזת שנקראה כולל המשתנה עצמו יש לספור בהמשך הקריאה. תכונה נבנית. |  |

למעשה, המונה מתעדכן בסריקה תחילית לעומק – כלומר, מצד אחד כאשר נכנסים לעומק העץ מעבירים לו את [abc]Before-ים למיניהם, וברגע שיוצאים חזרה מהעומק, מעדכנים חזרה את [abc]Next-ים. זה רלוונטי עבור S, בגלל ש-A,B כוללים אך ורק טרמינלי a ו-b בהתאמה – שם הדברים יותר פשוטים באופן משמעותי (צריך לשמור רק אורכים ו״כמה לדלג״ בהדפסה)

**שאלה 2:**

|  |  |
| --- | --- |
| משמעות התוכנה | שם התכונה |
| ערך מספרי של איבר בודד. |  |
| רשימת הערכים שמייצג ערך המשתנה. |  |
|  |

הערות: tail, cons, greater, sum, max כולן עושות על ערך רשימה את מה שהפעולה עצמה עושה כמוגדר במטלה – אדגיש שהן לא משנות את הרשימה, אלא מחזירות אחת חדשה עם השינויים. Copy מעתיקה רשימה ומחזירה אחת חדשה, Append מוסיפה איבר לרשימה קיימת. באופן טריוויאלי, כאשר מופעלת פונקציה על token כלשהו (טרמינל) – הכוונה היא להפעלה על ערכו הלקסיקלי.

**שאלה 4:**

1. הערה: אני מניח שכל המשתנים בתוכנית מוגדרים כint-ים (כלומר, זה רק חלק מתוכנית שמתחילה לפני כן. הוספתי halt כדי שזרימת התוכנית תהיה ברורה וודאית – היכן יש להמשיך אם בכלל). הערות לגבי התרגום הוספתי כ-// - C++ Styled, וברור שאינן חלק מהתרגום.
2. RLSS not\_geq j 5 // `j < 5` is like `not (j >=5)`
3. JMPZ not\_geq 5 // go to `while` if j >= 5
4. IASN z 17 // z = 17
5. JMP 9 // halt
6. RLSS continueloop z 30 // while (z<30)
7. JMPZ continueloop 9 // break if condition fails
8. IADD z z b // z += b
9. JMP 5 // possibly another iteration
10. HALT
11. על אותו משקל (מובן שניתן לעשות אופטימיזציה לקוד, אבל לא התבקשתי כאן)  
    אופציה ראשונה:
12. IMLT tmp z 10 // tmp:=z\*10
13. IEQL casejmp tmp 10 // check case: 10
14. JMPZ casejmp 6 // check next case
15. IASN c 100 // c = 100
16. JMP 8 // fall-through
17. IEQL casejmp tmp 20 // check case: 20
18. JMPZ casejmp 10 // next case (default)
19. IASN c 200 // c = 200
20. JMP 11 // break
21. IASN c 300 // default
22. HALT

אופציה שנייה:

1. IMLT tmp z 10 // tmp:=z\*10
2. INQL casejmp tmp 10 // check z\*10!=10
3. JMPZ 7 // execute case 10
4. INQL casejmp tmp 20 // check z\*10!=20
5. JMPZ 8 // execute case 20
6. JMP 10 // execute default
7. IASN c 100 // case: 10 (no break!)
8. IASN c 200 // case: 20
9. JMP 11 // break
10. IASN c 300 // case: default
11. HALT