♋ ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20364

קומפילציה

חוברת הקורס – סתיו 2025א

ערך: גדי פסח

אוקטובר 2024 - סמסטר סתיו- תשפ"ה

פנימי – לא להפצה.

כל הזכויות שמורות לאוניברסיטה הפתוחה. ©

תוכן העניינים

אל הסטודנטים

אנו מקדמים את פניכם בברכה עם הצטרפותכם אל הלומדים בקורס "קומפילציה".

הקורס "קומפילציה" הוא קורס מתקדם, אשר לימודו דורש בשלות בתחום מדעי המחשב, והכרות עם תחומים רבים שנתקלתם בהם במהלך לימודיכם. אנחנו מקווים שתמצאו עניין ברעיונות ובשיטות הנלמדים בקורס, גם אם הבנתם דורשת השקעה של זמן ומחשבה.

בחוברת זו תמצאו לוח זמנים ופעילויות, תנאים לקבלת נקודות זכות בקורס ואת המטלות.

אתם מוזמנים לפנות אליי בקשר לנושאים מנהליים, או בקשר לחומר הנלמד:

- **דואר אלקטרוני:** gadips@gmail.com.
- **פקס:** 09-7780605 (מזכירות מדעי המחשב).
- **טלפון ושעות קבלה :** יפרסמו יותר מאוחר באתר הבית של הקורס ותשלח הודעה בדואר.
- **דואר:**

גדי פסח, מדעי המחשב, האוניברסיטה
הפתוחה רבוצקי 108 ת.ד. 808
רעננה 43104

לקורס קיים אתר באינטרנט בו תמצאו חומרי למידה נוספים.
בנוסף, האתר מהווה עבורכם ערוץ תקשורת עם צוות ההוראה ועם סטודנטים אחרים בקורס. פרטים על למידה מתוקשבת ואתר הקורס, תמצאו באתר שה"ם בכתובת:
http://openu.ac.il/shoham

מידע על שירותי ספרייה ומקורות מידע שהאוניברסיטה מעמידה לרשותכם, תמצאו באתר הספריה באינטרנטwww.openu.ac.il/Library.

לתשומת לב הסטודנטים הלומדים בחו"ל:
למרות הריחוק הפיסי הגדול, נשתדל לשמור אתכם על קשרים הדוקים ולעמוד לרשותכם ככל האפשר.
הפרטים החיוניים על הקורס נכללים בחוברת הקורס וכן באתר הקורס.
מומלץ מאד להשתמש באתר הקורס ובכל אמצעי העזר שבו וכמובן לפנות אלינו במידת הצורך.

ניתן להיעזר במנחים בשעות ההנחיה הטלפונית שלהם או בשעת המפגשים.
תוכלו למצוא מידע מנהלי כללי בקטלוג הקורסים ובידיעון אקדמי. עדכונים יישלחו מדי סמסטר.

אני וצוות הקורס מאחלים לכם לימוד פורה ומהנה.
בברכה,
גדי פסח
מרכז ההוראה בקורס

לוח זמנים ופעילויות (מס' קורס /א2025)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	29.10.2024-01.11.2024	פרק 1		
2	03.11.2024-8.11.2024	פרק 3		
3	10.11.2024-15.11.2024	פרק 4		ממן 11 15.11.2024
4	17.11.2024-22.11.2024	פרק 4		
5	24.11.2024-29.11.2024	פרק 4		ממן 12 29.11.2024
6	01.12.2024-06.12.2024	פרק 4		
7	08.12.2024-13.12.2024	פרק 5		ממן 13 13.12.2024
8	15.12.2024-20.12.2024	פרק 5		

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*

לוח זמנים ופעילויות - המשך

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
9	22.12.2024-27.12.2024 (ה-ו חנוכה)	פרק 5		
10	29.12.2024-03.01.2025 (א-ה חנוכה)	פרק 6		ממן 14 03.01.2025
11	05.01.2025-10.01.2025	פרק 6		
12	12.01.2025-17.01.2025	פרק 7		ממן 15 17.01.2025
13	19.01.2025-24.01.2025	פרק 8		
14	26.01.2025-31.01.2025	פרק 9+8		
15	02.02.2025-03.02.2025	פרק 9+8		ממן 16 21.3.2025

מועדי בחינות הגמר יפורסמו בנפרד

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*

2. תאור המטלות

קראו היטב עמודים אלו לפני שתתחילו לענות על השאלות

פתרון המטלות הוא חלק בלתי נפרד מלימוד הקורס – מטרתן היא לתת לכם הערכה מהימנה על מידת הבנתכם ושליטתכם בחומר הקורס, ולהפנות את תשומת לבכם לנושאים חשובים בחומר הלימוד. המטלות יבדקו על-ידי המנחה ויוחזרו לכם בצירוף הערות המתייחסות לתשובות.

2.1 מבנה המטלות

המטלות בקורס הן משני סוגים: מטלות רגילות ומטלת פרוייקט (כל המטלות מכונות ממ"ן - ראשי תיבות של "מטלת מנחה").

מטלות רגילות (ממ"ן 11 – 15)
כל אחת מהמטלות הרגילות מכילה מספר שאלות עיוניות ורובן מכילות גם שאלות תכנות.

- **שאלות עיוניות**

מטרת השאלות העיוניות לעזור בהבנת חומר הלימוד וכן לבדוק את הבנתכם. סוג השאלות דומה לסוג השאלות שיופיע בבחינת הגמר.

שימו לב: מומלץ לפתור שאלות עיוניות רבות ככל האפשר, כי פתרון שאלות אלו מהווה הכנה טובה לקראת הבחינה.

פרויקט המהדר (ממ"ן 16)
הפרויקט הוא <u>מטלת חובה</u>.
פרויקט המהדר הוא פרוייקט תכנות רחב היקף, הכולל תכנון ומימוש של מהדר לשפת תכנות פשוטה שהוגדרה לצורך זה.

2.2 המטלות, נושאיהן וניקודן

המטלות הרגילות מלוות את חומר הלימוד בקורס, והפרויקט מתייחס במשולב לחלק ניכר מחומר הלימוד. לכל מטלה נקבע משקל; ניתן לצבור עד 30 נקודות. חובה להגיש מטלות במשקל של 24 נקודות לפחות.
שימו לב: ממ"ן 16 (פרוייקט המהדר) הוא מטלת חובה.

שימו לב:
ללא הגשת מטלת החובה וצבירת 24 נקודות לא ניתן יהיה לקבל נקודת זכות בקורס.

להלן פירוט המטלות, נושאי המטלות והניקוד לכל מטלה.

ממ"ן	פרקים בספר הלימוד	ניקוד
11	<u>פרק 3</u>	3
12	<u>פרקים 3,4</u> – שימוש ב-bison	3
13	<u>פרק 4</u>	3
14	<u>פרקים 5,6,7</u>	3
15	<u>פרקים 7,8,9</u>	3
16 חובה	פרוייקט המהדר	15 (חובה)

לתשומת לבכם!
כדי לעודדכם להגיש לבדיקה מספר רב של מטלות הנהגנו את ההקלה שלהלן:

אם הגשתם מטלות מעל למשקל המינימלי הנדרש בקורס, **המטלות** בציון הנמוך ביותר, שציוניהן נמוכים מציון הבחינה (**עד שתי מטלות**), לא יילקחו בחשבון בעת שקלול הציון הסופי.

זאת בתנאי שמטלות אלה **אינן חלק מדרישות החובה בקורס** ושהמשקל הצבור של המטלות האחרות שהוגשו, מגיע למינימום הנדרש.

זכרו! ציון סופי מחושב רק לסטודנטים שעברו את בחינת הגמר בציון 60 ומעלה והגישו מטלות כנדרש באותו קורס.

3. תנאים לקבלת נקודות זכות

1. הגשת מטלת החובה (הפרויקט).
2. צבירת 24 נקודות **לפחות** במטלות (15 נקודות בפרויקט ו- 9 נקודות לפחות במטלות הרגילות).
3. ציון של לפחות 60 נקודות בבחינת הגמר.
4. ציון סופי בקורס של 60 נקודות לפחות.

שאלון למטלת מנחה (ממ״ין)11

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 11**

משקל המטלה: **3 נקודות**

מספר השאלות: **3**

מס׳ הקורס: 20364	
מס׳ המטלה: 11	
מחזור: א 2025	

מועד משלוח המטלה: 15.11.2024

עבור כל תרגיל תכנות, נא להגיש את קובצי המקור שכתבתם, קובץ הרצה ודוגמא (אחת או יותר) לקובצי קלט ופלט מתאימים. הגישו גם קובץ README עם הסבר כיצד ניתן לבנות את קובץ ההרצה.

שאלה 1 (25%)

כתבו תכנית ב- flex שתדפיס את הקלט שלה עם השינויים הבאים:
כל מופע של סיפרה בודדת יוחלף בספרות רומיות: 1 יוחלף ב- I, 2 יוחלף ב- II, 3 ב- III, 4 ב- IV, 5 ב- V, 6 ב- VI, 7 ב- VII, 8 ב- VIII, ו- 9 יוחלף ב- IX. מספר הכולל יותר מסיפרה אחת יישאר ללא שינוי.

(סיפרה בודדת כאן היא כל סיפרה שאין לפניה או אחריה סיפרה נוספת).

למשל אם הקלט הוא

```
10 green bottles hanging on the wall
10 green bottles hanging on the wall
And if 1 green bottle should accidentally fall,
There'll be 9 green bottles hanging on the wall.
```

הפלט המבוקש הוא:

```
10 green bottles hanging on the wall
10 green bottles hanging on the wall
And if I green bottle should accidentally fall,
There'll be IX green bottles hanging on the wall.
```

התוכנית תקבל את קובץ הקלט כ- command line argument. בתור ברירת מחדל היא תקרא מה- standard input. את הפלט היא תכתוב ל- standard output.

שאלה 2 (תוכנית מחשב - 60%)

כתבו מנתח לקסיקלי (lexer) לשפת CPL. ראו הגדרות של האסימונים בממ"ן 16.

מומלץ להשתמש בכלי התוכנה flex או בכלי דומה (רבים משתמשים ב- sly או ב- ply

שהם כלים עבור python). אפשר לכתוב את התוכנית

באחת מהשפות C, C++, Java, Python . גם שפות נוספות ישקלו בחיוב.

הבהרות

יש לממש את המנתח הלקסיקלי כפונקציה (שתיקרא בהמשך על-ידי המנתח התחבירי).
הפונקציה תחזיר בכל קריאה את סוג "האסימון הבא" בקלט ואת ערכי תכונותיו (אם ישנן כאלה). את ערך התכונה אפשר "להחזיר" ע"י כתיבתו למשתנה גלובלי.
כדי לאפשר את בדיקת המנתח הלקסיקלי בשלב זה, הוסיפו תכנית ראשית, שתייצג עבור המנתח הלקסיקלי את "שאר הקומפיילר".

התכנית הראשית תקרא למנתח הלקסיקלי שוב ושוב עד לסוף קובץ הקלט ותייצר קובץ פלט עם תיאור האסימונים שנמצאו.

יש לקבוע אלו תכונות (אם בכלל) יש לאסימונים השונים. נזכיר שתכונה של אסימון מספקת מידע נוסף עבור האסימון מעבר לסוג שלו. לדוגמא טבעי שלאסימונים המייצגים אופרטורים (למשל ADDOP) תהיה תכונה המציינת את סוג האופרטור (פלוס או מינוס).
שימו לב שלחלק גדול מהאסימונים אין תכונות.

המנתח הלקסיקלי צריך גם לדלג על הערות (הערות בשפת CPL מתוארות בממן 16)
ועל white space (רווחים, טאבים ...)
הוא גם צריך לשמור את מספר השורה הנוכחית במשתנה גלובלי.

במקרה של גילוי שגיאה – יש להוציא הודעת שגיאה (ל- standard error) הכוללת את מספר השורה בה היא נפלה. אחרי גילוי שגיאה, ה- lexer ימשיך לקרוא את הקלט כדי לזהות אסימונים נוספים.

שימו לב שאין הרבה שגיאות שמנתח לקסיקלי אמור להבחין בהם.
למשל כל סוגי השגיאות הבאות <u>אינם</u> מענינו של המנתח הלקסיקלי:

-- חסרים אסימונים בקלט
-- יש אסימונים מיותרים בקלט
-- אסימונים מופיעים בקלט בסדר לא נכון

.

בשגיאות האלו אמור להבחין ה- parser שבו לא נעסוק בממן הזה.

שימו לב שאם אתם משתמשים ב- flex אז אין צורך להשתמש כאן ב- `start conditions` (אולי חוץ מהטיפול בהערות).

שימו לב שהחלוקה של האסימונים לקטגוריות keyword, symbols, operators (בממן 16) היא לצורך הנוחות. אבל כל keyword הוא אסימון מסוג אחר למשל יש אסימון מסוג WHILE ואסימון אחר מסוג IF. אין אסימון שנקרא KEYWORD.
דבר דומה נכון עבור הסימבולים והאופרטורים השונים: אין אסימון שנקרא SYMBOL -- יש אסימון מסוג נקודה פסיק, אסימון אחר מסוג פסיק וכן הלאה. כך גם אין אסימון שנקרא OPERATOR -- יש אסימונים מסוג ADDOP, MULOP או RELOP.

הממשק

קראו לתוכנית שלכם cla (קיצור של CPL Lexical Analyzer).

את התוכנית מפעילים משורת הפקודה של Windows כך:
cla <file_name>.ou

(ניתן להשתמש גם ב- Linux).

יש לכתוב שורת "חותמת" עם שם הסטודנט או סטודנטית ל- standard error וגם לקובץ הפלט.

הקלט של התכנית הוא קובץ טקסט (עם סיומת ou) המכיל תוכנית בשפת CPL.
הפלט הוא קובץ טקסט המכיל תיאור של האסימונים שהופיעו בקלט.
עבור כל אסימון כזה יופיע בפלט סוג האסימון, ה- lexeme (המחרוזת כפי שהופיעה בקלט) וערך התכונה שלו (אם יש לו).
התיאור של כל אסימון יופיע בשורה נפרדת. השם של קובץ הפלט יהיה זהה לשם של קובץ הקלט מלבד הסיומת שתהיה tok.

לדוגמא אם בקלט הופיע
foo = bar;
אז הפלט יכול להראות כך:

```
token        lexeme       attribute
-----        ------       ---------
ID             foo           foo
=              =
ID             bar           bar
SEMICOLON       ;
```

כאן (באופן שרירותי) לאסימון נקודה פסיק ניתן השם SEMICOLON בעוד שלאסימון '=' לא ניתן שם אלפבתי אלא הוא מופיע בתור '='. החליטו לבד אם אתם רוצים לתת שמות לאסימונים המייצגים סימבולים.

הערה: ה- attribute לעיתים קרובות זהה ל- lexeme אבל לא בהכרח. ה- lexeme הוא מחרוזת. ה- attribute לא בהכרח מיוצג בתכנית כמחרוזת. למשל ה- attribute של האסימון ID יכול להיות מצביע לכניסה המתאימה בטבלת הסמלים. (לצרכים שלנו זה בהחלט בסדר שהתכונה של ID תהיה זהה לשם של ה- ID (כלומר זהה ל- lexeme) התכנית תהיה פשוטה יותר אם ה- lexer לא יתעסק עם טבלת הסמלים).

<u>הגשה</u>
נא הגישו את הקובץ (או קבצים) שכתבתם, קובץ הרצה של התוכנית שלכם <u>ולפחות דוגמא אחת של קלט ופלט</u>. יש לצרף גם קובץ README שמסביר כיצד ניתן לבנות את קובץ ההרצה.

שאלה 3 (15%)

עבור הדקדוקים הבאים ענו על השאלות הבאות:

מהי שפת הדקדוק? יש לתת תיאור פורמלי ככל שניתן. אם יש דרך לרשום ביטוי רגולרי אז יש לכתוב את הביטוי.
האם הדקדוק חד-משמעי? אם לא אז הראו מילה (רצוי קצרה) בשפה שיש לה שני עצי גזירה שונים. אם התשובה היא כן אז הסבירו מדוע לכל מילה בשפה של הדקדוק יש עץ גזירה יחיד (אין צורך בהוכחה פורמלית).

```
1. S -> aSb | cS | Sc | z

2. T -> T T | a T b | b T a |epsilon
```

שאלון למטלת מנחה (ממ״ן)12

שם הקורס: קומפילציה	**מס׳ הקורס: 20364**
שם המטלה: ממ״ן 12	**מס׳ המטלה: 12**
משקל המטלה: 3 נקודות	**מחזור: א 2025**

מספר השאלות: 6

מועד משלוח המטלה: 29.11.2024

שאלה 1 (15%)

נתון דקדוק הבא המתאר ביטויים עם האופרטורים % ,! ,^:
(כאן הסוגריים, האופרטורים, INUM ו- FNUM הם הטרמינלים של הדקדוק)

```
S -> S^S | !S | S%S | (S) | INUM | FNUM
```

שימו לב ש-! הוא אופרטור אונארי. ^ ו- % הם אופרטורים בינאריים.

א. רשמו דקדוק חד משמעי השקול לדקדוק הנתון כך שהדקדוק ישקף את סדר העדיפויות הבא.

ל- ! תהיה העדיפות הגבוהה ביותר
ל- ^ תהיה עדיפות בינונית
ל- % תהיה העדיפות הנמוכה ביותר

לאופרטור ^ תהיה אסוציאטיביות ימנית. לאופרטור % תהיה אסוציאטיביות שמאלית.
האופרטור ! אינו אסוציאטיבי: זה לא חוקי (לפי הדקדוק) לכתוב למשל
`INT_NUM ! INT_NUM ! INT_NUM`

רמז: ראו דקדוק (4.1) בסעיף 4.1.2 בספר הלימוד.

ב. ציירו עץ גזירה עבור המחרוזת `INUM ! ^ (INUM) ^INUM`.
יש להשתמש בדקדוק שרשמתם בסעיף א.

שאלה 2 (10%)

סעיף א נתון הדקדוק הבא

המשתנים הם S, A, B. שאר הסימנים הם טרמינלים.

```
S -> Ag | Ac | B
A -> Sz | B
B -> ab
```

הפעילו על הדקדוק את האלגוריתם לסילוק רקורסיה שמאלית. ציינו במפורש מה הסדר שנקבע למשתנים. בתשובה יש לשמור על השמות המקוריים של המשתנים S, A, B ולא לשנות אותם למשל ל- A_1, A_2, A_3. זה יקל על קריאת הדקדוק החדש.

סעיף ב

הפעילו על הדקדוק הבא את האלגוריתם לצמצום גורמים שמאליים (left factoring).

```
S -> zA | zB | zAc
A -> ah | ahB
B -> b
```

שאלה 3 (20%)

נתון הדקדוק G. האותיות הגדולות הם המשתנים. האותיות הקטנות הן הטרמינלים.

```
(1) S -> G B A
(2) S -> c S
(3) A -> az
(4) B -> epsilon
(5) G -> g G
(6) G -> epsilon
```

א. חשבו את FIRST ו-FOLLOW לכל אחד ממשתני הדקדוק. אין צורך לפרט את מהלך החישוב.

ב. בנו טבלת פיסוק תחזית (טבלת LL(1)) לדקדוק G.

האם הדקדוק הינו דקדוק מסוג LL(1) ? שימו לב ששאלה זו שונה מהשאלה הבאה (שאינה נשאלת כאן): האם ניתן להפעיל על הדקדוק הנתון טרנספורמציות כך שיתקבל דקדוק שקול שהוא מסוג LL(1) ?

שאלה 4 (15%)

סעיף א

השלימו את הטבלה הבאה המתארת ריצה של `parser` המשתמש בטבלת ה- LL(1)

שבניתם בשאלה הקודמת על מחרוזת הקלט `gaz`.

בכל שלב ציינו בעמודה parser action את כלל הגזירה בו משתמשים או כתבו match.

(אם ה- parser מקבל את המילה אז ה- action האחרון יהיה accept).

האם המפסק (parser) מצליח לגזור את המילה?

parser stack	remaining input	parser action
S$	gaz$	
		predict S->GBA

סעיף ב

ציירו עץ גזירה של המילה gaz ומספרו את צמתי עץ הגזירה לפי הסדר בו הצמתים מגיעים לראש המחסנית במהלך ה- parsing של מילה זאת.
שורש העץ המסומן ב- S יקבל את המספר 1.

לאיזה סוג של מעבר על עץ הגזירה מתאים המספור של הצמתים ? התשובה צריכה להיות אחד מהבאים: preorder, inorder, postorder.

שאלה 5 (20%)

סעיף א יש לכתוב <u>recursive descent parser</u> עבור הדקדוק שמופיע בשתי השאלות הקודמות כלומר עבור כל משתנה של הדקדוק הגדירו את הפונקציה המתאימה. אפשר להשתמש בפסאודו קוד.
טבלת ה- LL(1) שבניתם עבור הדקדוק יכולה להקל על כתיבת ה- parser.

השתמשו בפונקציה match שפועלת כך (כאן הנחנו שכל סוג של אסימון מיוצג ע"י מספר שלם).

```
void match(int token) {
    if (lookahead == token())
        lookahead = lexer();
    else
        error(); // handle syntax error
}
```

(אין צורך לכתוב את הפונקציות lexer ו- error).

סעיף ב
מה יהיה רצף הקריאות לפונקציות כאשר ה- recursive descent parser עובד על הקלט gaz ?
נניח שבהתחלה קוראים לפונקציה S(). עבור כל קריאה לפונקציה שמוגדרת עבור משתנה של הדקדוק, רשמו מתי מתחילים לבצע אותה ומתי מסיימים.
עבור קריאות לפונקציה match רשמו רק את הארגומנט שמועבר לפונקציה.

כלומר יש לתאר את רצף הקריאות באופן הבא:

```
enter S();
enter G();
match(g);
```

```
...

exit S();
```

ציירו גם עץ גזירה של המילה gaz ומספרו את צמתי עץ הגזירה לפי הסדר בו ה- parser "מבקר" בצמתי העץ (שורש העץ המסומן במשתנה ההתחלתי יקבל את המספר 1) .
כאשר ה-parser עושה match לטרמנינל אז הוא "מבקר" בעלה המתאים בעץ (המסומן באותו טרמינל). כאשר ה- parser קורא לפונקציה עבור משתנה של הדקדוק אז הוא "מבקר" בצומת המתאים המסומן באותו משתנה. כאשר ה- parser משתמש בכלל אפסילון (כלל גזירה שבו מופיע אפסילון בגוף של הכלל) אז ה- parser "מבקר" בעלה המתאים המסומן באפסילון.

לאיזה סוג של מעבר על עץ הגזירה מתאים המספור של הצמתים ? התשובה צריכה להיות אחד מהבאים: preorder, inorder, postorder.

שאלה 6 (20%)

נתון הדקדוק הבא (הטרמינלים כאן הם 1,2,3):

```
(1) S -> 1A3
(2) S -> B2
(3) A -> 2A
(4) A -> epsilon
(5) B -> 1
(6) B -> epsilon
```

דקדוק זה אינו LL(1): בטבלת ה- LL(1) יש קונפליקט בין שני כללי הגזירה של S בעמודה 1.

בנו את טבלת ה- LL(2) של הדקדוק. האם הדקדוק הוא LL(2) ?
באתר הקורס ניתן למצוא הסבר כיצד בונים טבלת LL(2).

שאלון למטלת מנחה (ממ״ן)13

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 13**

משקל המטלה: **3 נקודות**

מספר השאלות: **4**

מועד משלוח המטלה: **13.12.2024**

מס׳ הקורס:	**20364**
מס׳ המטלה:	**13**
מחזור:	**א 2025**

שאלה 1 (25%)

נתון הדקדוק הבא שנקרא לו G. (המשתנים כתובים באותיות גדולות, הטרמינלים - באותיות קטנות).

```
1. S -> ABz
2. A -> Aa
3. A -> Bc
4. B -> b
5. B -> epsilon
```

א. בנו את אוטומט פריטי LR(0) של הדקדוק הנתון. כלומר, רשמו את האוסף של קבוצות פריטי LR(0), וציירו את קשתות המעברים בין הקבוצות. אל תשכחו לעבור קודם לדקדוק מורחב (אם כי זה לא באמת נחוץ עבור הדקדוק הנתון). באוטומט אמורים להופיע 9 מצבים.

ב. בנו את טבלת פיסוק SLR(1) לדקדוק G (action ו-goto). האם הדקדוק הוא SLR(1) ? אם לא – מה סוג הקונפליקט (או קונפליקטים) המופיעים בטבלה ?
הערה: נא לא לשנות את המספור של כללי הגזירה (כדי להקל על הבדיקה). אין צורך לתת מספר לכלל הגזירה S' -> S כי המספר של כלל זה לא יופיע בטבלה (במקום לעשות reduce לפי כלל זה עושים accept)

שאלה 2 (25%)

סעיף א . בנו את אוטומט פריטי LR(1) לדקדוק G הנ״ל, הפעם עם פריטי LR(1).
באוטומט אמורים להיות 10 מצבים.

סעיף ב האם בטבלת LR(1) של הדקדוק הנתון יש קונפליקטים ? במילים אחרות, האם הדקדוק הוא LR(1)? נמקו בקצרה.
<u>אין צורך לבנות את כל טבלת ה- LR(1)</u> אבל בנו (לצורך תרגול) את אחת השורות בטבלה.
אם יש בטבלה קונפליקט (או קונפליקטים) אז בנו שורה בה יש קונפליקט (הראו את כל השורה, לא רק את הכניסה בה מופיע קונפליקט). אם אין בטבלה קונפליקטים אז בנו רק את השורה בטבלה עבור המצב ההתחלתי.

סעיף ג . כמה מצבים יהיו בטבלת ה- LALR(1) של הדקדוק G ? האם G הוא דקדוק מסוג LALR(1)? נמקו. <u>אין צורך לבנות את טבלת ה- LALR(1)</u>.

שאלה 3 (15%)

נתונה טבלת SLR(1) של הדקדוק הבא

```
1. S -> c A B
```

```
2. S -> a
3. A -> a
4. B -> B b
5. B -> epsilon
```

	a	b	c	$	S	A	B
0	s3		s2		1		
1				accept			
2	s5					4	
3				r2			
4		r5		r5			6
5		r3		r3			
6		s7		r1			
7		r4		r4			

סעיף א הראו את שלבי הריצה של parser המשתמש בטבלה הנתונה על הקלט cab. יש להשלים את הטבלה הבאה:

parser stack	remaining input	parser action
0	cab$	

ה-action בכל שלב הוא shift או reduce או accept.
במקרה של reduce רשמו גם את כלל הגזירה בו משתמשים.

כשאתם רושמים את תוכן המחסנית -- נוח (לפחות עבור הבודק) לרשום את המצבים ואת סימני הדקדוק לסירוגין למרות שבפועל יש רק מצבים על המחסנית. למשל רשמו 0 a 7 b 12 c 25 במקום 0 7 12 25.

סעיף ב
ציירו עץ גזירה של המילה cab ומספרו את צמתי עץ הגזירה לפי הסדר בו ה- parser "מבקר" בצמתי העץ: כאשר ה-parser עושה shift לטרמינל הוא "מבקר" בעלה המתאים בעץ (המסומן באותו טרמינל). כאשר ה- parser עושה reduce למשתנה הוא "מבקר" בצומת המתאים המסומן באותו משתנה. כאשר ה- parser מצמצם לפי כלל אפסילון למשל A -> epsilon נאמר שהוא מבקר בעלה המתאים המסומן באפסילון ולאחר מכן הוא מבקר בצומת המסומן ב- A (ההורה של העלה המסומן באפסילון).

לאיזה סוג של מעבר על עץ הגזירה מתאים המספור של הצמתים? התשובה צריכה להיות אחד מהבאים: preorder, inorder, postorder.

שאלה 4 (35%)

בשאלה זו עליכם לבנות מפסק (parser) לשפה CPL -- השפה המוגדרת בממן 16.
אפשר לעשות זאת באופן ידני אבל קל יותר להשתמש ב- bison או ב- parser generator אחר.

כדי להפעיל את bison , עליכם ליצור עבורו קובץ קלט פשוט, המכיל רק את תיאור הדקדוק.
לדוגמה, לדקדוק של שפת CPL יש לבנות קובץ בעל הצורה הכללית הבאה :

```
%{
%}
%token ID
%token NUM
…
…     (Define all terminals in the grammar.
           Symbols such as +, -, (, ) don't need to be
defined).
%%
program    :      declarations  stmt_block ;
. . .

type       :     INT
           |    FLOAT
           ;
 . . .
assignment_stmt : ID '=' expression ';'
  … (Continue with all grammar productions).
```

- הקובץ יקרא cpl.y
- אתם צריכים להפעיל את bison בעזרת האופציה "v-", ייווצר קובץ נוסף בעל סיומת out, המכיל מידע על המפסק -- המצבים שלו (כל מצב הוא קבוצת פריטים) וה- actions וה- goto של כל מצב. <u>ודאו שאין הודעות שגיאה בקובץ cpl.output</u> bison לא אמור להוציא הודעות שגיאה על הדקדוק של ממן 16.

- שורת הפקודה היא: `bison -v  cpl.y`

בתור ברירת מחדל, bison בונה parser המשתמש בטבלת LALR(1). זה מספיק לצרכינו.
תוכנת bison המתאימה ל- Windows נמצאת (גם) באתר הקורס בתיקיה "כלי תוכנה".

שימו לב שבשאלה זו אין צורך להריץ את ה- parser -- רק לתת ל- bison לייצר אותו.
לכן גם אין צורך כאן במנתח לקסיקלי ואין צורך לכתוב ()main.

<u>הגשה</u>
יש להגיש את הקבצים cpl.y ו- cpl.out.

5 דצמבר 2024 תוקן מספור הכללים בשאלה 3 (והטבלה תוקנה בהתאם)

שאלון למטלת מנחה (ממ״ין)14

שם הקורס: קומפילציה	**מס׳ הקורס: 20364**
שם המטלה: ממ״ן 14	**מס׳ המטלה: 14**
משקל המטלה: 3 נקודות	**מחזור: א 2025**

מספר השאלות: 3

מועד משלוח המטלה: 3.1.2025 (נדחה ל- 10.1.2025)

שאלה 1 (25%)

בשני הסעיפים אתם מתבקשים לכתוב סכימת תרגום (translation scheme) (להבדיל מהגדרה מונחת תחביר Syntax Directed Definition).
נזכיר שבסכימת תרגום ניתן לשלב פעולות סמנטיות (actions : קטעי קוד מוקפים בסוגריים מסולסלים) בצד ימין של כללי הגזירה. המיקום של ה- action קובע מתי הוא יתבצע בעת המעבר על עץ הגזירה.
אם יש בכך צורך, מותר להוסיף לדקדוק את כלל הגזירה `S' -> S` כאשר S' הוא המשתנה ההתחלתי החדש.

<u>סעיף א</u>
נתון הדקדוק הבא :

```
S -> aSb
S -> cc
```

כתבו סכימת תרגום שתכתוב לפלט תיאור של עץ הגזירה של הקלט.
למשל עץ הגזירה של המילה aaccbb נראה כך :

```
      S
    / | \
   a  S  b
     /|\
    a S b
      / \
      c c
```

והתיאור של העץ שיכתב לפלט הוא :

```
S
  a
  S
    a
    S
      c
      c
    b
  b
```

בתיאור העץ, האינדנטציה (הזחה) של סימן מראה את מרחקו מהשורש של עץ הגזירה. בדוגמא ה- S הראשון מופיע עם אינדנטציה 0, ה- S השני עם אינדנטציה 1 והשלישי עם אינדנטציה 2. שני סימני ה- c מופיעים עם אינדנטציה 3.

בסכימת התרגום יש להשתמש בפונקציה print(symbol, indent) כדי לכתוב את הסימנים לפלט. הפלט בדוגמא מתקבל ע"י סדרת הקריאות הבאה (כל קריאה ל- print כותבת גם newline):

```
print(S, 0)
print(a, 1)
print(S, 1)
print(a, 2)
print(S, 2)
print(c, 3)
print(c, 3)
print(b, 2)
print(b, 1)
```

אפשר לפתור את השאלה תוך שימוש במשתנה גלובלי. קראו למשתנה indent. (צריך גם לאתחל אותו כנדרש).
לחילופין ניתן להשתמש בתכונה מורשת כדי לייצג את האינדנטציה.

<u>סעיף ב</u>

נתון הדקדוק הבא:

```
S -> cSaS
S -> bA
S -> cB
A -> Aa
A -> epsilon
B -> bB
B -> b
```

הוסיפו לדקדוק הנתון פעולות סמנטיות כך שבפלט יופיעו סימני ה- a שמקיימים את התנאי הבא: מספר סימני ה- a שהופיעו בינתיים (כשקוראים את הקלט משמאל לימין) גדול ממספר סימני ה- b שהופיעו בינתיים.
למשל עבור הקלט ccbbbabaaaaa הפלט יהיה aa כי רק שני ה- a - ים האחרונים עומדים בתנאי.

בפתרון יש להשתמש בתכונה מורשת אחת לפחות (מותרות גם תכונות נבנות). <u>אין להשתמש במשתנים גלובליים</u> (כדי לתרגל שימוש בתכונות).

תנו הסבר קצר על כל תכונה וציינו אם היא מורשת או נבנית (inherited או synthesized)

שאלה 2 (25%)

נתון דקדוק המתאר שפה של "ביטויי רשימות". התוצאה של כל ביטוי היא מספר טבעי (גדול או שווה לאפס) או רשימה של מספרים טבעיים.

מרכיבי השפה הם:

- **רשימות פשוטות של מספרים טבעיים** - לדוגמה: [12 , 4 , 100 , 57]
מספר יכול לחזור יותר מפעם אחת ברשימה. הערה: רשימות ריקות של מספרים עשויות להיווצר כתוצאה של הפעלה של הפעולות divide ו- tail.

- **פעולת equal (list)** - מקבלת רשימה list ומחזירה 1 אם כל איברי הרשימה שווים ואחרת מחזירה 0. לדוגמה:
```
equal([12 , 4 , 100]) = 0
equal([7 , 7 , 7]) = 1
```
שימו לב שפעולה זו מחזירה מספר ולא רשימה.
במקרה שהרשימה ריקה אז equal תחזיר 1.

- **פעולת sum (list)** - מקבלת רשימה list ומחזירה את סכום המספרים ברשימה.
לדוגמה: `sum([12 , 4 , 100]) = 116`
שימו לב שפעולה זו מחזירה מספר ולא רשימה.
במקרה שהרשימה ריקה אז sum תחזיר אפס.

- **פעולת append(n, list)** - מקבלת מספר n ורשימה list, ומחזירה רשימה שבה n מצורף בתור איבר אחרון ל-list .
לדוגמה:
```
append(5, [1, 4, 100]) = [1 , 4 , 100, 5]
```

- **פעולת tail(list)** – מקבלת רשימה list ומחזירה רשימה שבה נמחק האיבר הראשון של הרשימה list. אם list היא הרשימה הריקה אז tail מחזירה רשימה ריקה.
לדוגמא `tail([35, 100, 17]) = [100, 17]`

- **- פעולת divide(n, list)** - מקבלת מספר n ורשימה list, ומחזירה רשימה הכוללת את כל המספרים ב- list שמתחלקים ב- n.
לדוגמה:
```
divide(5, [12 , 35, 4 , 100]) = [35, 100]
```

דוגמה לביטוי בשפה:
```
sum(append(equal([3, 3]),
           divide(10, [30, 8, 50])))
```

התוצאה של הביטוי היא המספר 81 (סכום המספרים ברשימה [30, 50, 1])

הנה דקדוק המתאר את שפת הביטויים (כאן התוצאה של הביטוי יהיה מספר. תתי ביטויים עשויים להיות רשימות של מספרים).

1. S → ITEM
2. L → **[** ITEMLIST **]**
3. L → **TAIL** (L **)**
4. L → **APPEND (** ITEM **,** L **)**
5. L → **DIVIDE (** ITEM **,** L **)**
6. ITEMLIST → ITEMLIST **,** ITEM
7. ITEMLIST → ITEM
8. ITEM → **SUM (** L **)**
9. ITEM -> **EQUAL (**L**)**
10. ITEM -> **NUMBER**

כאן האסימונים (למשל SUM) כתובים באותיות גדולות מודגשות. בקלט הם כתובים באותיות קטנות (למשל sum). גם הסוגריים (עגולות ומרובעות) והפסיקים הם אסימונים.

<u>כתבו סכימת תרגום המסתמכת על הדקדוק הנתון, אשר מדפיסה את התוצאה של חישוב הביטוי</u>.

<u>הערות:</u>

- מותר להגדיר תכונות למשתני הדקדוק לפי הצורך. אין להגדיר משתנים גלובאליים (כדי לתרגל שימוש בתכונות).
- הפעולות שמוצמדות לכל כלל צריכות להתייחס רק לתכונות של סימני הדקדוק המופיעים באותו כלל (שזה נכון לכל סכימת תרגום)
- האסימון **NUMBER** מייצג מספרים. הניחו שהתכונה NUMBER.val מציינת את ערך המספר.

- הפתרון צריך להיות high level. השתמשו בפסאודו קוד. אין להכנס לפירוט כיצד מיוצגות רשימות.

מותר להשתמש בפונקציות כמו למשל tail (list). אם המשמעות של הפונקציה אינה מובנת מאליה אז צרפו הסבר קצר.
על שאלה זאת ניתן לענות בלי לכתוב לולאות שעוברות על כל איברי הרשימה - הפונקציות שקוראים להן יעשו את זה (בשאלה הבאה תצטרכו לממש את הפונקציות האלו).

שאלה 3 (תוכנית מחשב - 50%)

יש לממש את סכימת התרגום שכתבתם בשאלה 2.
ניתן להשתמש ב- flex & bison או בכלים אחרים (למשל sly).
הקלט לתוכנית יהיה קובץ המכיל "ביטוי רשימה" כפי שתואר בשאלה 2.
קובץ הקלט ינתן כ- command line argument.
הפלט יהיה הערך של הביטוי. הוא יכתב ל- standard output.

- קראו לקובץ ההרצה list.exe. (או list אם אתם עובדים על Linux).

26 דצמבר 2024: תוקנה טעות בשאלה 2 (כלל גזירה 7 תוקן)

שאלון למטלת מנחה (ממ״ן)15

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 15**

משקל המטלה: **3 נקודות**

מספר השאלות: **6**

מס׳ הקורס: 20364	
מס׳ המטלה: 15	
מחזור: א 2025	

מועד משלוח המטלה: 17.1.2025 (נדחה ל- 24.1.2025)

שאלה 1 (9%)

נתונה תוכנית בשפה התומכת בקינון של פונקציות:

```
void  main() {
    void f() {
        int stam; // local to f
        void foo() {

            int bar() { ... }
            ...
        } /* end of foo */
        ...
    } /* end of f */

    void r() {...}

    ...
} /* end of main */
```

סעיף א. **(5%)** נניח שברגע מסוים בעת ביצוע התוכנית סדר הקריאות הוא כזה:

`main -> r -> f -> foo -> foo -> bar`

(אף אחת מהקריאות עוד לא חזרה). ציירו את תוכן המחסנית של רשומות ההפעלה (activation records a.k.a. stack frames) ברגע זה.

יש לציין עבור כל רשומת הפעלה לאיזה פונקציה היא שייכת ויש לצייר את מצביעי הגישה (access links a.k.a. static links). ציירו גם את ה-dynamic links. אין צורך לפרט את המבנה הפנימי של רשומות ההפעלה.

סעיף ב. **(4%)** בפונקציה bar מופיעה הפקודה `stam = 42;`. כאן stam הוא משתנה מקומי של הפונקציה f. הראו תרגום של stam = 42 לקוד ביניים (Three Address Code).

נניח שיש משתנה arp (קיצור של activation record pointer) שמצביע לשדה access link ברשומת ההפעלה של הפונקציה שרצה כרגע. כל access link מצביע ל- access link ברשומת ההפעלה המתאימה. המשתנה stam נמצא ב- offset 12 יחסית לשדה ה- access link ברשומת ההפעלה של f.

קוד הביניים כולל גם פקודות מסוג

```
x = *y
y[3] = 100
```

המשמעות של פקודות אלו דומה למשמעות שלהם בשפת C. (בשתי הפקודות הערך של y זו כתובת).

שאלה 2 (20%)

השאלה עוסקת בתרגום לולאות for_range לשפת Quad. (זו השפה שמשתמשים בה בממ"ן 16).

הנה כלל הגזירה:

```
stmt -> FOR_RANGE '(' ID '=' expression1 TO expression2
                                    ';' STEP NUM ')' stmt1
```

כאן FOR_RANGE, TO, STEP הם אסימונים המייצגים את המילים השמורות המתאימות (for_range, to, step). NUM הוא אסימון המייצג מספרים שלמים.
יש לו תכונה NUM.val המציינת את המספר. לאסימון ID יש תכונה ID.name המציינת את שם המשתנה.

המשמעות של המשפט היא:
שני הביטויים מחושבים (פעם אחת). נסמן את תוצאות שני הביטויים ב- r1, r2.
"משתנה הלולאה" ID מאותחל לערך r1. כל עוד הערך של משתנה הלולאה קטן או שווה ל- r2 ממשיכים לבצע את הלולאה כאשר בכל איטרציה מבצעים את גוף הלולאה (זה $stmt_1$) ולאחר מכן מוסיפים למשתנה ID את הערך NUM.val.

במילים אחרות, באיטרציה הראשונה הערך של משתנה הלולאה יהיה r1. באיטרציה השניה הוא יהיה r1+NUM.val. בשלישית הוא יהיה -- r1+2*NUM.val וכן הלאה. (אפשר להניח שאין כתיבה למשתנה הלולאה בגוף הלולאה).

במקרה ש- r1 > r2 אז גוף הלולאה לא יבוצע בכלל.

דוגמא:

```
for_range (foo = a+b to  (bar+17-c); step 4)
          z += foo;
```

אז אם נניח שהערך ההתחלתי של z הוא אפס, הערך של a+b זה 20 והערך של bar+17-c זה 30 אז גוף הלולאה (z+= foo;) יבוצע 3 פעמים כאשר משתנה הלולאה foo יקבל את הערכים (משמאל לימין) 20, 24, 28, 32. הערך הסופי של z יהיה 0+20+24+28 = 72

<u>הראו כיצד ניתן לתרגם משפטי for_range לשפת Quad</u>.

עליכם להוסיף פעולות סמנטיות (semantic actions) לכלל הגזירה הנ"ל. הפעולות הסמנטיות האלו (שיכתבו בפסאודו קוד) יכתבו לפלט קוד בשפת Quad.

ניתן להשתמש בפונקציות הבאות:
הפונקציה gen(code-to-print) כותבת את הקוד לפלט. (אפשר לקרוא לה print).
הפונקציה newlabel() מייצרת תוויות סימבוליות חדשות L1, L2 וכן הלאה.
הפונקציה label(label-name) מדפיסה תווית לפלט (ואחריה נקודותיים).
הפונקציה newtemp() מייצרת משתנים זמניים חדשים t1, t2 וכן הלאה.

אפשר להוסיף לדקדוק משתנה (או משתנים) שגוזרים את המילה הריקה (ולשייך להם action או תכונות)

הניחו שבזמן המעבר על ביטוי (expression) נוצר קוד עבור הביטוי. קוד זה מחשב את תוצאת הביטוי וכותב אותה למשתנה זמני. הסבירו באיזה תכונה (או תכונות) של expression אתם משתמשים.

שימו לב שלא נדרש כאן לכתוב actions ליצירת קוד עבור ביטויים. כללי הגזירה של expression לא נתונים בשאלה ואין צורך לכתוב אותם.

בשאלה זו מותר להניח שקוד Quad כולל תוויות סימבוליות אליהם ניתן לקפוץ.
(לאמיתו של דבר יעדים של קפיצות בשפת Quad הם מספרים סידוריים של פקודות).

שאלה 3 (20%)

ממשו ע"י recursive descent parser את סכימת התרגום ליצור קוד עבור משפטי for_range שכתבתם בשאלה 2. יש לכתוב רק את החלק הרלוונטי של הפונקציה stmt:

```
void stmt(){
     switch(lookahead){
        case WHILE: ...
        case ID: ...
        case IF:
        case FOR_RANGE: /* complete this code */

       ...
}
```

כתבו פסאודו קוד (אין צורך לכתוב תוכנית עובדת).

ניתן להשתמש בפונקציות שהוזכרו בשאלה 2 (gen, newlabel וכ"ו).

הניחו שהפונקציה expression (גם היא חלק מה- parser) מייצרת קוד עבור ביטוי במהלך ה- parsing שלו. קוד זה מחשב את הביטוי וכותב את התוצאה שלו לתוך משתנה. הפונקציה expression מחזירה את המשתנה הזה.

אתם לא צריכים לכתוב את הפונקציה expression.

שאלה 4 (20%)

השאלה עוסקת בקוד המיועד למכונת מחסנית (stack machine). במכונה זאת פקודות מתיחסות לאופרנדים הנמצאים על המחסנית ומאחסנות את התוצאה על המחסנית.

למען הפשטות נניח שכל הערכים הם מטיפוס int.
הנה הפקודות בהן (או בחלקן) נשתמש.
(1) פקודות המפעילות אופרטור אריתמטי בינארי. האופרטור מופעל על שני האופרנדים שבראש המחסנית כאשר האופרנד הימני בראש המחסנית והאופרטור השמאלי מתחתיו. לשני האופרנדים עושים pop ואת התוצאה דוחפים למחסנית במקומם.
הפקודות הן add, sub, mul, div (חיבור, חיסור, כפל וחילוק).

(2) פקודות השוואה המפעילות אופרטור השוואה בינארי. האופרטור מופעל על שני האופרנדים שבראש המחסנית והתוצאה נדחפת במקומם למחסנית. (גם כאן האופרנד הימני בראש המחסנית והאופרנד השמאלי מתחתיו). הערך שנדחף למחסנית הוא 1 (שמייצג את הערך true) או 0 (שמייצג את הערך false).
פקודות ההשוואה הן
eq (קיצור של equal)
ne (not equal)
gt (greater than)
lt (less than)
ge (greater or equal)
le (less or equal).

(3) פקודת קפיצה בלתי מותנית: jump label
(4) פקודות קפיצה עם תנאי: jump_if_zero label קופצת ל- label אם הערך בראש המחסנית הוא אפס. בכל מקרה עושים pop לערך שבראש המחסנית.
בדומה לכך הפקודה jump_if_not_zero label מבצעת קפיצה אם הערך שבראש המחסנית שונה מאפס. (ועושים pop לערך שבראש המחסנית).
(5) הפקודה store variable עושה pop לערך שבראש המחסנית ומאחסנת אותו במשתנה variable.
(6) הפקודה load variable דוחפת את הערך של המשתנה variable לראש המחסנית.
(ניתן להשתמש בקבוע מספרי במקום במשתנה לדוגמא load 7 דוחף 7 לראש המחסנית).
(7) pop מוחקת את המספר שבראש המחסנית (עושה pop למחסנית)
(8) dup (קיצור של duplicate) דוחפת למחסנית עותק נוסף של המספר שנמצא בראש המחסנית.
אם לדוגמא תוכן המחסנית היא 5 4 3 (5 בראש המחסנית) אז בעקבות dup התוכן יהיה 5 5 4 3.

דוגמא.
את הביטוי a+7*b ניתן לתרגם לקוד הבא:

```
load a
load 7
load b
mul
add
```

באופן כללי, האפקט הסופי היחיד של הקוד עבור ביטוי יהיה דחיפה של תוצאת הביטוי לראש המחסנית. במהלך חישוב הביטוי המחסנית עשויה לגדול ולקטון אבל הערכים שהיו על המחסנית לפני חישוב הביטוי ישארו ללא שינוי.

דוגמא
נתבונן במשפט

```
while (a < 3) {
    b = b + c;
    a = a + 1;
}
```

ניתן לתרגם את המשפט כך:

```
label1:   load a
          load 3
          lt
          jump_if_zero label2
          load b
          load c
```

```
            add
            store b
            load a
            load 1
            add
            store a
            jump label1
label2:
```

באופן כללי, הקוד עבור משפט אמור להשאיר את תוכן המחסנית ללא שינוי יחסית לתוכן לפני ביצוע המשפט. כמובן שבמהלך ביצוע המשפט המחסנית עשויה לגדול ולקטון.

תארו כיצד ניתן לייצר קוד (של מכונת מחסנית) עבור משפטי for_range (שתוארו בשאלה 2). לצורך כך הוסיפו פעולות סמנטיות ליצור קוד עבור מכונת מחסנית לכלל הגזירה הבא (זה אותו הכלל שהופיע בשאלה 2):

```
stmt -> FOR_RANGE '(' ID '=' expression1 TO expression2
                                    ';' STEP NUM ')' stmt1
```

הפעולות הסמנטיות ייצרו קוד המיועד למכונת מחסנית.
ניתן להשתמש באותן פונקציות כמו בשאלה 2 (gen, newlabel, label). אין כאן צורך בפונקציה newtemp.

הניחו שבזמן המעבר על expression מיוצר קוד לחישוב הביטוי.
במהלך חישוב הביטוי המחסנית עשויה לגדול ולקטון אבל ערכים שהיו על המחסנית לפני החישוב ישארו שם ללא שינוי. עם סיום חישוב הביטוי, ה- side effect היחיד יהיה שתוצאת החישוב תמצא בראש המחסנית.

(אינכם צריכים לטפל ביצור קוד עבור ביטויים כאן. אנו מניחים שיש כללי גזירה ו- actions שדואגים לייצר קוד עבור הביטויים).

שאלה 5 (15%)

סעיף א (5%) . נתון מערך דו מימדי A :

```
double A[20][30]
```

נניח שגודל של double בזיכרון הוא 8 בתים (bytes).

נתון שהאיבר הראשון במערך A[0][0] נמצא בכתובת 100 בזכרון.
מה הכתובת של A[10][5] בהנחה שהמערך נשמר לפי שורות ?
מה הכתובת בהנחה שהמערך נשמר לפי עמודות ?

סעיף ב (5%) כתבו type expression עבור הטיפוס של הפונקציה הנתונה bar.

```
struct  node {
      double num[3];
      struct node *next
      struct node *previous;
};

 struct node *bar(char *p,  int stam);
```

הערה: ל- type constructor עבור "רשומות" אפשר לקרוא record או struct.

כדי לכתוב type expression עבור טיפוס רקורסיבי (כמו struct node כאן) ניתן לכתוב משהו כזה:

A = struct (… pointer(A) …)

בהמשך ניתן להשתמש בשם A בתוך type expressions אחרים.

יש במדריך הלמידה דוגמאות ל- type expressions.

סעיף ג (5%)

ציירו AST (Abstract Syntax Tree) שמייצג את המשפט הבא:

```
for_range (foo = a+b to(bar-17-c); step 4)
          z += foo;
```

שאלה 6 (16%)

סעיף א. (8%)

שאלה זו עוסקת באלגוריתם של Baker ל- Garbage Collection.

נניח שהאובייקטים A, B, C, D, E, F כרגע מוקצים (allocated).

האובייקטים מכילים מצביעים (או references) לאובייקטים אחרים לפי הפרוט הבא:

A מכיל מצביעים ל- B, E
B מכיל מצביעים ל C, E
C מכיל מצביע ל- A
D מכיל מצביעים ל- E
E מכיל ל- A
F מכיל מצביע לעצמו

נניח שה- root set כולל מצביע ל- A. נניח גם שכאשר "סורקים" אוביקט המכיל מצביעים למספר אוביקטים אז אוביקטים אלו "מתגלים" לפי סדר אלפביתי.

האלגוריתם של Baker עושה שימוש בארבע רשימות:

unreached, unscanned, scanned free -1.

<u>השלימו את הטבלה הבאה</u>. בכל שורה אמורים לראות את תוכן שלוש הרשימות באחד השלבים של ריצת האלגוריתם. ההבדל בין שורות עוקבות בטבלה הוא שאחד האוביקטים הועבר מרשימה אחת לרשימה אחרת. למשל בשורה השניה, האוביקט A הועבר מהרשימה unreached לרשימה unscanned.

השלימו את הטבלה הבאה:

unreached	unscanned	Scanned
A B C D E F		
B C D E F	A	

(אם רשימה לא משתנה בצעד מסוים ניתן לרשום: ללא שינוי).

הניחו שהרשימה unscanned מנוהלת כתור. (בריישום תוכן הרשימה – האוביקט השמאלי ביותר הוא בראש התור).

מי הם האוביקטים שיועברו לרשימה free ?

סעיף ב (8%)

סעיף זה עוסק באלגוריתם של Cheney ל- Garbage Collection.

נניח שה- allocated objects (שכרגע נמצאים ב- From Space) הם כמו בסעיף א. ה- root set כולל מצביע ל- A.

נניח עוד שכל אוביקט תופס 100 בתים ושכאשר "סורקים" אוביקט אז האוביקטים שהוא מצביע אליהם "מתגלים" לפי סדר אלפביתי. עוד נניח שה- To Space מתחיל בכתובת 10, 000.

מה יהיו הכתובות של האוביקטים שיועתקו ל- To Space?

השלימו את הטבלה הבאה:

אוביקט	כתובת
A	
B	
C	
D	
E	
F	

שאלון למטלת מנחה (ממ״ן)16

שם הקורס: קומפילציה	**מס׳ הקורס: 20364**
שם המטלה: ממ״ן 16	**מס׳ המטלה: 16**
משקל המטלה: 15 נקודות	**מחזור: א 2025**

מספר השאלות: 1

מועד משלוח המטלה: 21.3.2025

שאלה 1 (100%)

1. פרוייקט המהדר

בפרוייקט זה עליכם לתכנן ולממש חלק קדמי של מהדר, המתרגם תוכניות משפת המקור CPL לשפה Quad.

שפת המקור CPL (Compiler Project Language) היא שפה דמוית פסקל או C, אך מוגבלת מהן בהרבה.

שפת הביניים Quad היא שפת פשוטה. השפות תוגדרנה בסוף המטלה.

2. תיאור פעולת המהדר

2.1. מה עושה המהדר?

המהדר יבצע את כל שלבי ההידור (החלק הקדמי) כפי שנלמדו בקורס, החל בניתוח לקסיקלי, דרך ניתוח תחבירי ובדיקות סמנטיות, ועד לייצור קוד ביניים בשפת Quad.

המהדר יקבל קובץ קלט המכיל תוכנית בשפת CPL. כפלט, ייצר המהדר קובץ המכיל תוכנית בשפת Quad. תוכלו להריץ את תוכניות ה-Quad הנוצרות בעזרת מפרש (interpreter) שנמצא באתר הקורס.

2.2. הממשק

המהדר יהיה תוכנית המופעלת משורת הפקודה של Windows.

שמו של המהדר הוא cpq (קיצור של CPL to Quad).

קובץ הריצה צריך להיקרא cpq.exe.

הקובץ עם הפונקציה הראשית של המהדר (main) צריך להיקרא cpq.c.

<u>קלט</u> – המהדר מקבל כפרמטר יחיד שם של קובץ קלט (קובץ טקסט המכיל תוכנית בשפת CPL).

הסיומת של שם קובץ הקלט צריכה להיות ou.

שורת הפקודה היא: cpq <file_name>.ou

<u>פלט</u> – המהדר יוצר קובץ טקסט עם שם זהה לשם קובץ הקלט ועם סיומת qud.

קובץ זה מכיל את תוכנית ה-Quad שנוצרה.

<u>טיפול בשגיאות ממשק</u> – במקרה של שגיאה בפרמטר הקלט, בפתיחת קבצים וכדומה, יש לסיים את הביצוע בצירוף הודעת שגיאה מתאימה למסך (stderr). במקרה כזה אין לייצר קובץ פלט.

כחלק מהטיפול בשגיאות ממשק, יש לוודא שהסיומת של קובץ הקלט היא נכונה.

שורת חותמת – יש לכתוב שורת "חותמת" עם שם הסטודנט, אשר תופיע במקומות הבאים:
ב-standard error
בקובץ ה-quad – אחרי הוראת ה-HALT האחרונה, וזאת כדי לא להפריע למפרש של שפת Quad.

2.3. טיפול בשגיאות

ייתכן שתוכנית הקלט תכיל שגיאות מסוגים שונים:
שגיאות לקסיקליות
שגיאות תחביריות
שגיאות סמנטיות

שימו לב:
במקרה של קלט המכיל שגיאה (מכל סוג שהוא) אין לייצר קובץ qud (גם לא קובץ qud ריק).
לאחר זיהוי של שגיאה לקסיקלית, תחבירית או סמנטית, יש להמשיך בהידור מהנקודה שאחרי השגיאה. זאת כדי לגלות שגיאות נוספות אם ישנן.
את הודעות השגיאה יש לכתוב ל- standard error. הודעת השגיאה צריכה לכלול את מספר השורה בה נפלה השגיאה.

3. מימוש המהדר

3.1. שימוש בכלים flex ו-bison

מומלץ להשתמש בכלי תוכנה flex & bison או בכלים דומים (לדוגמא ply או sly שהם כלים בשפת python).

flex הוא כלי אשר מייצר באופן אוטומטי מנתחים לקסיקליים. bison הוא כלי לייצור אוטומטי של מנתחים תחביריים.

ניתן לכתוב את הקומפיילר באחת מהשפות C. C++, Java, Python. מי שרוצה להשתמש בשפה אחרת מתבקש לפנות למנחה.

3.2 מבנה כללי

הקומפיילר יכול לבצע בדיקות סמנטיות וליצר את קוד ה- Quad כבר במהלך הניתוח התחבירי. זה אפשרי כי שפת CPL היא שפה פשוטה.
לחילופין ניתן לארגן את פעולת הקומפיילר באופן הבא:
המנתח התחבירי יצור `Abstract Syntax Tree (AST)` שמייצג את התוכנית המקורית.
ואז ניתן לייצר את קוד ה- Quad במעבר על העץ. את הבדיקות הסמנטיות ניתן לבצע בשלבים שונים: במהלך בנית העץ, במעבר נפרד על העץ או בזמן שמייצרים את קוד הביניים.

3.3 חישוב יעדי קפיצה

בקוד ה- Quad שמייצר המהדר עשויות להופיע פקודות JUMP או JMPZ, כאשר יעד הקפיצה הוא מספר שורה. לצורך חישוב יעדי הקפיצה, ייתכן שתבחרו להשתמש בהטלאה לאחור (backpatching), או בשיטה של ייצור קוד זמני המכיל תוויות סימבוליות (מחרוזות), ומעבר נוסף על הקוד כדי להחליף את התוויות הסימבוליות במספרי שורות.
לצורך מימוש השיטה שבה תבחרו תוכלו להחליט להחזיק בזיכרון את כל הקוד המיוצר, או שתוכלו לייצר קבצים זמניים, שבהם ייכתב הקוד בשלבי הביניים של הייצור. בדרך כלל האפשרות הראשונה פשוטה יותר.

3.3. מבני נתונים

במימוש המבנים שגודלם תלוי בקלט יש להעדיף הקצאת זיכרון דינמית על-פני הקצאה סטטית שגודלה חסום ונקבע מראש.
במימוש המבנים שגודלם קבוע וידוע מראש עדיפה כמובן הקצאה סטטית. במבנים אלה יש גם להעדיף מימוש "מונחה טבלה" , שבו מאוחסן המידע ב"טבלה" נפרדת, והקוד משמש לגישה לטבלה ולקריאתה.

מימוש טבלת הסמלים צריך לאפשר חיפוש מהיר. לכן אין להסתפק במימוש ע"י חיפוש ברשימה מקושרת שכוללת את כל הסמלים.

באופן דומה, אם אתם שומרים את פקודות ה- Quad הנוצרות ברשימה מקושרת אז יש להימנע מסריקות של הרשימה כדי למצוא את סופה כי פעולה זו (שמן הסתם תבוצע פעמים רבות) עלולה להאט את הקומפיילר באופן משמעותי. במקום זה ניתן ביחד עם כל רשימה כזאת להחזיק גם מצביע לאיבר האחרון שלה.

מותר להשתמש במבני נתונים המוגדרים בספריות (סטנדרטיות או לא סטנדרטיות) . מותר להשתמש בקוד שמממש מבני נתונים שנמצא באינטרנט אבל יש לתת קרדיט למקור.

3.4. סגנון תכנות

התוכנית שתכתבו צריכה לעמוד בכל הקריטריונים הידועים של תוכנית כתובה היטב: קריאות, מודולריות, תיעוד וכו'.

4. כיצד להגיש את הפרוייקט

4.1. תיעוד

יש לכתוב תיעוד בגוף התוכנית, כמקובל. תיעוד זה נועד להקל על קוראי התוכנית.
התיעוד צריך להבהיר קטעי קוד שאינם ברורים. עדיף לא להסביר בהערה מה שקל להבין מהקוד עצמו.

בנוסף, יש לכתוב **תיעוד נלווה**: מסמך נפרד, שאותו ניתן לקרוא באופן עצמאי, ללא קריאת התוכנית עצמה.

לתיעוד הנלווה שתי מטרות עיקריות: הסברים על שיקולי המימוש, ותיאור מבנה הקוד. יש להציג דיון ענייני בשיקולי המימוש.

אין צורך להכביר מילים. די בעמוד אחד או שניים של תיעוד.

4.2. מה להגיש

תיקיה src עם הקבצים שכתבתם.
קובץ הרצה
קובץ README עם הוראות לבנית קובץ ההרצה. אפשר להגיש גם makefile
תיעוד

4.3. בדיקת התכנית לפני ההגשה

מומלץ להשתמש במפרש של שפת Quad שנמצא באתר הקורס. בעזרתו תוכלו להריץ את תוכניות ה-Quad שיצרתם וכך לבדוק את תקינות הקוד המיוצר. כמו כן, תוכלו להיעזר בו כדי להבין את שפת Quad – תוכלו לכתוב תוכניות דוגמה קטנות בשפת Quad, ולהריץ אותן במפרש.

בנוסף לקלטים תקינים, נסו להריץ את הקומפיילר שלכם על תכניות קלט עם שגיאות (לקסיקליות, תחביריות וסמנטיות), כולל תוכניות המכילות יותר משגיאה אחת.

שפת המקור – שפת התכנות CPL
(Compiler Project Language)

1. מבנה לקסיקלי

בשפה CPL מוגדרים האסימונים הבאים:

אסימונים המייצגים מילים שמורות:

```
break case default else float if input int output switch while
```

אסימונים המייצגים "סימבולים":

```
(   )   {   }
,   :   ; =
```

אסימונים המייצגים אופרטורים:

```
RELOP:          ==  |  != | <  |  >  | >= | <=
ADDOP:          +   |  -
MULOP:          *   |  /
OR:             ||
AND:            &&
NOT:            !
CAST:         cast<int>    cast<float>
```

אסימונים נוספים:

```
ID:             letter (letter|digit)*
NUM:            digit+  | digit+.digit*

Where:  (Note: digit and letter are not tokens)
   digit:  0  |  1  | ... |  9
   letter: a  |  b  | ... |  z  |  A  |  B  | ... | Z
```

<u>הבהרות:</u>

1. בין האסימונים יכולים להופיע תווי רווח (space), תווי טאב (\t) או תווים המסמנים שורה חדשה (\n).
   תווים כאלה חייבים להופיע כאשר הם נחוצים לצורך הפרדה בין אסימונים (למשל, בין מלה שמורה לבין מזהה). בשאר המקרים, האסימונים יכולים להיות צמודים זה לזה, ללא רווח.
2. הערות בתוכנית מופיעות בין הגבולות /* .... */ (כמו בשפת C). אין קינון של הערות.
3. השפה היא case sensitive.

2. מבנה תחבירי:

Grammar for the programming language CPL

```
program -> declarations stmt_block

declarations -> declarations declaration
              | epsilon

declaration -> idlist ':' type ';'

type -> INT
        | FLOAT

idlist -> idlist ',' ID
          | ID

stmt -> assignment_stmt
      | input_stmt
      | output_stmt
      | if_stmt
      | while_stmt
      | switch_stmt
      | break_stmt
      | stmt_block


assignment_stmt -> ID '=' expression ';'

input_stmt -> INPUT '(' ID ')' ';'
output_stmt -> OUTPUT '(' expression ')' ';'

if_stmt -> IF '(' boolexpr ')' stmt ELSE stmt

while_stmt -> WHILE '(' boolexpr ')' stmt

switch_stmt -> SWITCH '(' expression ')' '{' caselist
                           DEFAULT ':' stmtlist '}'
caselist -> caselist CASE NUM ':' stmtlist
          | epsilon

break_stmt -> BREAK ';'

stmt_block -> '{' stmtlist '}'

stmtlist  ->  stmtlist stmt
               | epsilon

boolexpr -> boolexpr OR boolterm
         |   boolterm

boolterm -> boolterm AND boolfactor
         |   boolfactor

boolfactor ->  NOT '(' boolexpr ')'
            | expression  RELOP  expression

expression -> expression ADDOP  term
             | term
```

```
term -> term MULOP factor
        | factor

factor -> '(' expression ')'
          | CAST '(' expression ')'
          | ID
          | NUM
```

3. סמנטיקה

קבועים מספריים שאין בהם נקודה עשרונית הם מטיפוס int. אחרת הם מטיפוס float.

כאשר לפחות אחד האופרנדים של אופרטור בינארי אריתמטי (פלוס, מינוס ...) הוא מטיפוס float אז התוצאה של הפעלת האופרטור היא מטיפוס float ואחרת (כלומר שני האופרנדים מטיפוס int) התוצאה היא מטיפוס int.

כאשר אופרטור בינארי מופעל על אופרנדים מטיפוסים שונים , האחד מטיפוס int והשני מטיפוס float אז האופרנד מטיפוס int עובר המרה לערך מטיפוס float לפני הפעלת האופרטור.

יש להגדיר כל משתנה פעם אחת.

חילוק בין שני שלמים נותן את המנה השלמה שלהם.

פעולת השמה היא חוקית כאשר שני אגפיה הם מאותו טיפוס או שהאגף השמאלי הוא float. במקרה של השמה של ערך מסוג int למשתנה מסוג float, הערך מומר ל- float.

משפט break יכול להופיע רק בתוך לולאת while או בתוך switch. המשמעות שלו כמו בשפת C. בהיעדר break בקוד עבור case (במשפט switch) אז אחרי ביצוע הקוד עבור ה- case ממשיכים ומבצעים את הקוד עבור ה- case הבא (כמו בשפת C).

הביטוי שמופיע אחרי switch חייב להיות בעל טיפוס int. כך גם כל מספר שמופיע אחרי case.

האופרטור `cast<int>` עושה casting ל- `int`. האופרטור `cast<float>` עושה casting ל- `float`.

4. תוכנית לדוגמה:

```
/* Finding minimum between two numbers */
a, b: float;

{
    input(a);
    input(b);

    if (a < b)
       output(a);
    else
       output(b);
}
```

שפת המטרה – Quad

לכל הוראה בשפת Quad יש בין אפס לבין שלושה אופרנדים. תוכנית היא סדרה של הוראות בשפה. הפורמט המחייב של תוכנית הוא:

- הוראה אחת בכל שורה – סוג ההוראה (ה- opcode) כתוב תמיד **באותיות גדולות**.
- קוד ההוראה והאופרנדים מופרדים על ידי תו רווח אחד לפחות.
- בכל תוכנית מופיעה ההוראה HALT לפחות פעם אחת, בשורה האחרונה.

ישנם שלושה סוגי אופרנדים להוראות השפה:

1. **משתנים**. שמות המשתנים יכולים להכיל **אותיות קטנות, ספרות ו/או קו תחתון _**. (השם אינו יכול להתחיל בספרה).
2. **קבועים מספריים** (מטיפוס שלם או ממשי) הגדרתם זהה להגדרתם בשפת CPL.
3. **יעדי קפיצה:** נרשמים כמספר שלם המסמן מספר סידורי של הוראה בתכנית (החל מ-1).

למשתנים ולקבועים בשפת Quad יש טיפוס - שלם או ממשי.
אין הכרזות של משתנים. השימוש הראשון במשתנה קובע את הטיפוס שלו.
למשל `IASN foo 3` קובע שהטיפוס של foo הוא שלם. `RASN foo 7.5` קובע שהטיפוס שלו הוא ממשי. טיפוס של משתנה איננו יכול להתחלף במהלך התוכנית. ישנן הוראות שונות עבור שלמים ועבור ממשיים. אין לערבב בין הטיפוסים. קיימות גם שתי הוראות המאפשרות מעבר בין שלמים וממשיים.

בשפה אין משתנים בולאניים, הוראות השוואה מחשבות מספר: 1 עבור True ו- 0 עבור False.
כמו כן קיימת הוראת קפיצה בלתי מותנית והוראת קפיצה מותנית. (המבצעת למעשה הוראת "if not … goto …").

הוראות שפת Quad

בטבלה הבאה:

A מציין משתנה שלם

B ו- C מציינים משתנים שלמים או קבועים שלמים

D מציין משתנה ממשי

E ו- F מציינים משתנים ממשיים או קבועים ממשיים.

L מציין יעד קפיצה (מספר שורה).

<u>שימו לב:</u> A, B, C, D, E, F הם סימנים מופשטים, שיכולים לציין משתנה כלשהו. המשתנים המופיעים בפועל בתוכנית צריכים להיכתב באותיות קטנות (מותרים גם ספרות וקו תחתון).

Opcode	Arguments	Description
IASN	A B	A := B
IPRT	B	Print the value of B
IINP	A	Read an integer into A
IEQL	A B C	If B=C then A:=1 else A:=0
INQL	A B C	If B<>C then A:=1 else A:=0
ILSS	A B C	If B<C then A:=1 else A:=0
IGRT	A B C	If B>C then A:=1 else A:=0
IADD	A B C	A:=B+C
ISUB	A B C	A:=B-C
IMLT	A B C	A:=B*C
IDIV	A B C	A:=B/C

RASN	D E	D := E
RPRT	E	Print the value of E
RINP	D	Read a real into D
REQL	A E F	If E=F then A:=1 else A:=0
RNQL	A E F	If E<>F then A:=1 else A:=0
RLSS	A E F	If E<F then A:=1 else A:=0
RGRT	A E F	If E>F then A:=1 else A:=0
RADD	D E F	D:=E+F
RSUB	D E F	D:=E-F
RMLT	D E F	D:=E*F
RDIV	D E F	D:=E/F

ITOR	D B	D:= real(B)
RTOI	A E	A:= integer(E)

JUMP	L	Jump to Instruction number L
JMPZ	L A	If A=0 then jump to instruction number L else continue.

HALT		Stop immediately.

דוגמא

הנה תוכנית בשפת CPL

```
/* Finding minimum between two numbers */
a, b: float;

{
    input(a);
    input(b);

    if (a < b)
       output(a);
    else
       output(b);
}
```

הנה תרגום אפשרי לשפת Quad

RINP a
RINP b
RLSS less a b
JMPZ 7 less
RPRT a
JMP 8
RPRT b
HALT

בהצלחה