ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20364

קומפילציה

חוברת הקורס – סתיו 2021א

ערך : גדי פסח

אוקטובר 2020 - סמסטר סתיו- תשפ"א

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:** gadipe@openu.ac.il.
- **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 באינטרנט www.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

1. לוח זמנים ופעילויות (20364/א2021)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	23.10.2020-18.10.2020	פרק 1	מפגש ראשון פרקים 1+3	
2	30.10.2020-25.10.2020	פרק 3		
3	06.11.2020-01.11.2020	פרק 4	מפגש שני פרק 4 חלק א	ממ"ן 11 1.11.2020
4	13.11.2020-08.11.2020	פרק 4		
5	20.11.2020-15.11.2020	פרק 4	מפגש שלישי פרק 4 חלק ב	ממ"ן 12 20.11.2020
6	27.11.2020-22.11.2020	פרק 4		
7	04.12.2020-29.11.2020	פרק 5	מפגש רביעי פרק 5	ממ"ן 13 4.12.2020
8	11.12.2020-06.12.2020 (ו חנוכה)	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	18.12.2020-13.12.2020 (א-ו חנוכה)	פרק 5	מפגש חמישי פרק 6	
10	25.12.2020-20.12.2020	פרק 6		ממ"ן 14 25.12.2020
11	01.01.2021-27.12.2020	פרק 6	מפגש שישי פרק 7	
12	08.01.2021-03.01.2021	פרק 7		ממ"ן 15 8.1.2021
13	15.01.2021-10.01.2021	פרק 8	מפגש שביעי פרקים 9+8	
14	22.01.2021-17.01.2021	פרק 9+8		ממ"ן 16 19.2.2021

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.

פרוייקט המהדר (ממ"ן 16)

הפרוייקט הוא מטלת חובה.

פרוייקט 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, והפרוייקט 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	פרק 3	3
12	פרקים 3,4 – שימוש ב-bison	3
13	פרק 4	3
14	פרקים 5,6,7	3
15	פרקים 7,8,9	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״ן)11

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 11**

משקל המטלה: **3 נקודות**

מספר השאלות: **3**

מועד משלוח המטלה: 1.11.2020

מס׳ הקורס: 20364	
מס׳ המטלה: 11	
מחזור: א 2021	

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

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 שורה שמספרה אי זוגי תמוספר (השורה הראשונה מספרה 1). 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:

```
1.   10 green bottles hanging on the wall
     10 green bottles hanging on the wall
3.   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 (60%)

יש לכתוב מנתח לקסיקלי המזהה אסימונים בקלט בו מופיע מידע על שיאי עולם בריצת 100 מטר לגברים.

הנה דוגמא לקלט:

World Record World Record

```
[1] <time> 9.86 <athlete> "Carl Lewis" <country> "United
States" <date> 25 August 1991

[2] <time> 9.69  <athlete> "Tyson Gay" <country> "United
States" <date>  20 September 2009

[3] <time> 9.82  <athlete> "Donovan Baily"  <country>
"Canada" <date> 27 July 1996


[4] <time> 9.58
   <athlete> "Usain Bolt"
   <country> "Jamaica"  <date> 16 August 2009

[5] <time> 9.79 <athlete> "Maurice Greene"  <country>
"United State"  <date> 16 June 1999
```

עליכם לקבוע בעצמכם את סוגי האסימונים ואת ההגדרות המדויקות של ה- lexemes המתאימים לכל סוג. כל הגדרה סבירה תתקבל.

למשל ניתן לקבוע ש- <athlete> בקלט מזוהה כאסימון אחד מסוג ATHLETE. לחילופין אפשר לקבוע ש- <athlete> יזוהה כשלושה אסימונים שניתן לקרוא להם LEFT_BRACKET, ATHLETE, RIGHT_BRACKET.

הפלט לתוכנית יכיל עבור כל אסימון שנמצא בקלט את סוג האסימון, ה- lexeme (המחרוזת כפי שהופיעה בקלט) וערך התכונה או תכונות של האסימון (אם יש כאלו).
הערך של התכונה של האסימון (שנקרא גם ערך סמנטי של האסימון) מספק מידע נוסף על האסימון מעבר לסוג שלו. שימו לב שלחלק מהאסימונים אין צורך להגדיר תכונות כי סוג האסימון כבר מספק את כל המידע אודותיו (למשל האסימון ATHLETE הנ"ל).

לדוגמא עבור הקלט שהופיע למעלה השורות הראשונות של הפלט עשויות להראות כך:

TOKEN	LEXEME	ATTRIBUTE
TITLE	WORLD RECORD	
TITLE	WORLD RECORD	
NUMBER	[1]	1

המנתח הלקסיקלי בדרך כלל מדלג על white space (סימני רווח, טאבים, newlines) אבל יש מקרים בהם white space יכול להחשב חלק מאסימון למשל חלק מאסימון המייצג שמות של אצנים.

כאשר המנתח הלקסיקלי נתקל בשגיאה יש להוציא הודעת שגיאה (עם מספר השורה שבה נפלה השגיאה) ולהמשיך לנתח את האסימונים הבאים. שימו לב שאין הרבה שגיאות שמנתח לקסיקלי אמור להבחין בהם.
למשל כל סוגי השגיאות הבאות אינם מענינו של המנתח הלקסיקלי:

- חסרים אסימונים בקלט (למשל חסר שם של אצן)
- יש אסימונים מיותרים בקלט (למשל מופיע אצן עם שני שמות)
- 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.

מומלץ לכתוב את התוכנית בעזרת כלי התוכנה flex או כלי אחר דומה לו.
אפשר להשתמש בשפות C, C++, JAVA, python. מי שרוצה להשתמש בשפת תכנות אחרת מתבקש לפנות קודם למנחה.

מומלץ לעשות הפרדה בין שתי המטלות: זיהוי אסימונים והדפסת הפלט.
כלומר המנתח הלקסיקלי לא צריך בעצמו לכתוב לפלט אלא הוא צריך לספק מספיק מידע (סוג האסימון, ערך סמנטי ...) כדי שמי שקורא לו יוכל לבצע את הכתיבה לפלט.

הממשק

התוכנית תקרא athlete.exe

היא תופעל משורת הפקודה של Windows (או Linux).
התוכנית תקבל שם של קובץ קלט כארגומנט.
שורת הפקודה היא: `athlete <file_name>`

הפלט יכתב ל- standard output

נא לצרף לפחות דוגמא אחת של קובץ קלט וקובץ עם הפלט המתאים.

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 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 -> cSa | cSb | Sa | Sb | H
   H -> H h | epsilon

2. S -> S a S b | S b S 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 2021**

מספר השאלות: 5

מועד משלוח המטלה: 20.11.2020

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

שאלה 1 (תוכנית מחשב - 50%)

כתבו מנתח לקסיקלי לשפת CPL. ראו הגדרות של האסימונים בממ״ן 16.

מומלץ להשתמש בכלי התוכנה flex או בכלי דומה. אפשר לכתוב את התוכנית באחת מהשפות C, C++, Java, Python . גם שפות נוספות ישקלו בחיוב.

<u>הבהרות</u>

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 תאור האסימונים שנמצאו.

יש לקבוע אלו תכונות (אם בכלל) יש לאסימונים השונים.

המנתח הלקסיקלי צריך גם לדעת לדלג על הערות (הערות בשפת CPL מתוארות בממן 16).
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.

שימו לב שהחלוקה של האסימונים לקטגוריות keyword, symbols, operators (בממן 16) היא לצורך הנוחות. אבל כל keyword הוא אסימון מסוג אחר כלומר יש אסימונים IF, WHILE וכן הלאה. דבר דומה נכון עבור הסימבולים והאופרטורים השונים.

<u>הממשק</u>

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

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

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

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

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

שאלה 2 (10%)

נתון דקדוק הבא המתאר ביטויים עם האופרטורים +, #, ^:

```
S -> S^S | S+S | S#S | (S) | NUM | ID
```

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. לשאר האופרטורים תהיה אסוציאטיביות שמאלית.

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

ב. ציירו עץ גזירה עבור המחרוזת `NUM#(ID+ID)^NUM#NUM`.
יש להשתמש בדקדוק שרשמתם בסעיף א.

שאלה 3 (10%)

נתון הדקדוק הבא

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

```
S ->  c | cA
A -> Bb | B000
B -> A4 | b
```

הפעילו על הדקדוק את האלגוריתם לסילוק רקורסיה שמאלית. ציינו במפורש מה הסדר שנקבע למשתנים.
הפעילו על הדקדוק שהתקבל את האלגוריתם לצמצום גורמים שמאליים (left factoring).

שאלה 4 (15%)

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

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

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 5 (15%)

נתון הדקדוק הבא:

```
1. S -> AB
2. S -> Cz
3. A -> abC
4. A -> epsilon
5. B -> bC
6. C -> cbC
7. C -> epsilon
```

השלימו את הטבלה הבאה המתארת ריצה של LL(1) parser על מחרוזת הקלט bcb.

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

האם המפסק מצליח לגזור את המילה?

Parser stack	Remaining input	Parser action
S$	bcb$	predict S->AB

יש לבנות את טבלת הפיסוק לדקדוק ואז להשתמש בה.

שאלון למטלת מנחה (ממ״ן)13

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 13**

משקל המטלה: **3 נקודות**

מספר השאלות: **3**

מועד משלוח המטלה: **4.12.2020**

מס׳ הקורס: 20364	
מס׳ המטלה: 13	
מחזור: א 2021	

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

שאלה 1 (40%)

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

```
1. S -> ABc
2. S -> GB
3. A -> Aa
4. A -> epsilon
5. B -> b
6. G -> aG
7. G -> b
```

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.

ב. בנו את טבלת פיסוק SLR(1) לדקדוק G (action ו-goto). שימו לב שיש לפחות קונפליקט אחד בטבלה (הדקדוק אינו SLR(1)).

ג. נתונה טבלת SLR(1) של הדקדוק הבא

```
1. S -> AB
2. A -> Ca
3. A -> z
4. B -> b
5. C -> C c
6. C -> epsilon
```

	a	b	c	z	$	S	A	B	C
0	r6		r6	s4		1	2		3
1					accept				
2		s6						5	
3	s7		s8						
4		r3							
5					r1				
6					r4				
7		r2							
8	r5		r5						

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.

ד. בנו את אוטומט פריטי LR(1) לדקדוק G (כמו בסעיף א', הפעם עם פריטי LR(1)).
באוטומט אמורים להיות 13 מצבים. **שימו לב שחזרנו לדקדוק של סעיף א.**
האם בטבלת LR(1) של הדקדוק הנתון יש קונפליקטים ? במילים אחרות, האם הדקדוק הוא LR(1)? נמקו בקצרה. אין צורך לבנות את טבלת ה- LR(1).

ה. כמה מצבים יהיו בטבלת ה- LALR(1) של הדקדוק G ? האם G הוא דקדוק מסוג LALR(1)? נמקו. גם במקרה זה אין צורך לבנות את כל טבלת ה- LALR(1).

שאלה 2 (30%)

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.
- שורת הפקודה היא: `bison -v cpl.y`

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

bison לא אמור להוציא הודעות שגיאה על הדקדוק של ממן 16.

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

<u>הגשה</u>

יש להגיש את הקבצים cpl.y ו- cpl.out.

שאלה 3 (10%)

נתון הדקדוק הבא שהוא LL(1).

יש לכתוב recursive descent parser עבור דקדוק זה. אפשר להשתמש בפסאודו קוד.

נוח להכין טבלת LL(1) עבור הדקדוק לפני כתיבת ה- parser. (במקרה זה קל להסתדר בלי זה בגלל שהדקדוק פשוט. בכל אופן טבלה עבור הדקדוק הזה התבקשה כבר בשאלה 5 בממן הקודם).

```
1. S -> AB
2. S -> Cz
3. A -> abC
4. A -> epsilon
5. B -> bC
6. C -> cbC
7. C -> epsilon
```

שאלה 4 (20%)

נתון הדקדוק הבא:

```
S -> 0A0 | B0 | 1
A -> 2A | 0 | epsilon
B -> 1B | 2
```

א. נמקו בקצרה מדוע הדקדוק אינו LL(1).
ב. בנו את טבלת LL(2) של הדקדוק. האם הדקדוק הוא LL(2)?

שאלון למטלת מנחה (ממ״ן)14

שם הקורס: קומפילציה	**מס׳ הקורס: 20364**
שם המטלה: ממ״ן 14	**מס׳ המטלה: 14**
משקל המטלה: 3 נקודות	**מחזור: א 2021**

מספר השאלות: 4

מועד משלוח המטלה: 25.12.2020

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

A

נתון הדקדוק הבא:

```
S -> ScAB
S -> bSc
S -> a
A -> aA
A -> a
B -> Bb
B -> epsilon
```

בשני הסעיפים אתם מתבקשים לכתוב סכימת תרגום (translation scheme).
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.

c
א. (5%) הוסיפו לדקדוק זה פעולות סמנטיות כך שבמהלך קריאת הקלט, כל סימן a יודפס פעמיים. דוגמה: עבור הקלט acab הפלט יהיה aaaa.

ב. (15%) הוסיפו לדקדוק זה פעולות סמנטיות כך שרק סימני a המקיימים את שני התנאים הבאים יודפסו:

תנאי ראשון: יש לכל היותר 7 סימנים לפני סימן ה- a
תנאי שני: אין סימן b מיד לפני סימן ה- a

אילוץ נוסף הוא שבמקרה שאורך מילת הקלט קטן מ- 10 אין להדפיס שום סימן.

לדוגמא עבור הקלט bbbbacaaaabcccc הפלט יהיה aa כי רק סימני ה- a השני והשלישי (משמאל) עומדים בתנאים. עבור הקלט acab לא יודפס דבר כי אורך המילה קטן מ- 10.

יש להשתמש בתכונה מורשת אחת לפחות (מותרות גם תכונות נבנות).

שאלה 2 (25%)

הדקדוק הבא מייצר מחרוזות שמתארות עצים.

1. S -> tree
2. tree -> **SUM_ODD (** treelist **)**
3. tree -> **SUM_EVEN (** treelist **)**
4. tree -> **SIZE (** treelist **)**
5. tree -> **IGNORE (** treelist **)**
6. tree -> **NUMBER**
7. treelist -> treelist tree
8. treelist -> tree

בעלים של העצים מופיעים מספרים טבעיים.
צמתים פנימיים (עם ילד אחד או יותר) מסומנים בתווית (sum_odd או sum_even או size או ignore). עץ הכולל צומת אחד בלבד (שחייב להיות עלה) מתואר ע"י המספר שמופיע בעלה. עץ שיש בו יותר מצומת אחד מתואר ע"י התווית שמופיעה בשורש ולאחריה סוגריים שבתוכם מופיעות מחרוזות (מופרדות ע"י white space) המתארות את תתי העצים של השורש.

למשל המחרוזת

```
sum_odd (sum_even (sum_even(4 size(7))) 5)
         3)
```

מתארת את העץ הבא.

```
          sum_odd
         /       \
      sum_even    5
         |
      sum_even
     /     \
    4      size
             |
             7
```

נגדיר שלכל עץ (או תת עץ) יש ערך שהוא מספר טבעי באופן הבא:
הערך של עץ המורכב מצומת אחד בלבד (שחייב להיות עלה) הוא המספר המופיע בעלה.

הערך של עץ ששורשו מסומן ב- sum_even הוא סכום הערכים של תתי העצים שמתחת לשורש שערכם זוגי. (אם אין תתי עצים כאלו הערך הוא אפס). הערך של עץ ששורשו מסומן ב- sum_odd מוגדר באופן דומה כאשר רק הערכים האי זוגיים נלקחים בחשבון. הערך של עץ ששורשו מסומן ב- size הוא מספר הצמתים בעץ. הסימון ignore בצומת משמעותו שיש

להתעלם מהצומת ומכל הצמתים מתחתיו כאילו הם אינם קיימים. זו שגיאה לסמן את השורש של העץ כולו ב- ignore.

לדוגמא נתבונן בעץ הבא. כאן לשורש המסומן ב- sum_odd יש שלושה ילדים שבעצמם מהווים שורשים של שלושה תתי עצים. מתת העץ הימני נתעלם (בגלל ה- ignore).
הערך של תת העץ השמאלי הוא 1. הערך של תת העץ האמצעי הוא 2+6=8.
הערך של העץ כולו הוא 1 (1 הוא מספר אי זוגי בעוד ש- 8 הוא זוגי).

```
      sum_odd
     /     \      \
   1   sum_even ignore
        /   \       |
       2     6     1201
```

דוגמא נוספת: הערך של העץ הראשון שצויר למעלה הוא 5.
הערך של תת העץ ששורש סומן ב- size הוא 2. ה- sum_even מעליו ערכו 4+2 = 6
וזה גם ערכו של ה- sum_even מעליו (הילד של השורש של העץ כולו).

<u>כתבו סכימת תרגום המסתמכת על הדקדוק הנתון אשר מדפיסה את הערך של העץ המופיע בקלט.</u>

<u>הערות:</u>

מותר להגדיר תכונות למשתני הדקדוק לפי הצורך. אין להגדיר משתנים גלובאליים.
הפעולות שמוצמדות לכל כלל גזירה צריכות להתייחס רק לתכונות של הסימנים המופיעים באותו כלל ולא למשתנים גלובליים.
האסימון **NUMBER** מייצג מספרים. הניחו שהתכונה NUMBER.val מציינת את ערך המספר.

בפתרון כל התכונות צריכות להיות מספריות. אין צורך להגדיר תכונה שערכה הוא עץ.

שאלה 3 (תוכנית מחשב - 45%)

יש לממש את סכימת התרגום שכתבתם בשאלה 2.
ניתן להשתמש ב- flex & bison או בכלים אחרים.
הקלט לתוכנית יהיה קובץ המכיל תאור של עץ כפי שתואר בשאלה 2.
קובץ הקלט ינתן כ- command line argument.
הפלט יהיה הערך של העץ. הוא יכתב ל- standard output.

קראו לקובץ ההרצה tree.exe.

שאלה 4 (10%)

כתבו את המשפטים הבאים בשפת QUAD (השפה שמוגדרת בממן 16).
א.

```
for(j=0;j<10;j++)
    if (a>10)
        bar = 2 * bar;
```

(אין צורך לבצע כאן אופטימיזציות -- הכוונה לתרגום ישיר של הקוד לשפת Quad).
ב.

```
switch (10*k-1) {
       case 19: a = a + 7; break;
       case 99: a = a – 4; break;
       default: a = 0;
}
```

יש להניח שכל המשתנים הם מטיפוס int

שאלון למטלת מנחה (ממ״ן)15

שם הקורס: **קומפילציה**

שם המטלה: **ממ״ן 15**

משקל המטלה: **3 נקודות**

מספר השאלות: **7**

מועד משלוח המטלה: **8.1.2021**

מס׳ הקורס: 20364	
מס׳ המטלה: 15	
מחזור: א 2021	

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

שאלה 1 (12%)

נתונה תוכנית בשפה התומכת בקינון של פונקציות:

```
void  main() {
    int foo;
    void f() { ... }
    void g() {
        int h() { ... }
        ...
    } /* end of g */
        ...
} /* end of main */
```

סעיף א. **(6%)** נניח שברגע מסוים בעת ביצוע התוכנית סדר הקריאות הוא כזה:

main → g → f → g → h → f

(אף אחת מהקריאות עוד לא חזרה). ציירו את תוכן המחסנית של רשומות ההפעלה (activation records) ברגע זה.
יש לציין עבור כל רשומת הפעלה לאיזה פונקציה היא שייכת ויש לצייר את מצביעי הגישה (access links a.k.a. static links). אין צורך לפרט את המבנה הפנימי של רשומות ההפעלה.

סעיף ב. **(6%)** בפונקציה h מופיעה הפקודה `foo = 17;`. כאן foo הוא משתנה מקומי של ()main. הראו תרגום של foo = 17 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 foo נמצא ב- offset 12 יחסית לשדה ה- access link ברשומת ההפעלה של main.

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 (12%)

השאלה עוסקת בתרגום משפטי strange_if לשפת Quad. (זו השפה שמשתמשים בה בממ"ן 16).
הנה כלל הגזירה:

```
stmt -> STRANGE_IF stmt '(' boolexp ')' stmt
```

המשמעות של המשפט (המטופש) הזה דומה למשמעות של משפט if רגיל: אם התנאי הבוליאני מתקיים מתבצע המשפט הראשון ואחרת מתבצע המשפט השני.

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

עליכם להוסיף פעולות סמנטיות (semantic actions) לכלל הגזירה הנ"ל כלומר מדובר בחלק מסכימת תרגום (translation scheme). הפעולות הסמנטיות (שיכתבו בפסאודו קוד) ידפיסו 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 boolexp נוצר קוד עבור הביטוי הבוליאני. קוד זה מחשב את תוצאת הביטוי וכותב אותה למשתנה זמני. (לחילופין ניתן להשתמש בקוד "עם קפיצות": זה קוד שקופץ למקום אחד בתוכנית אם התנאי מתקיים ולמקום אחר אם אינו מתקיים). הסבירו באיזה תכונות של boolexp אתם משתמשים.

שימו לב שלא נדרש כאן לכתוב actions ליצירת קוד עבור ביטויים בוליאניים. כללי הגזירה של boolexp 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 (12%)

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

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

. . .
}

כתבו פסאודו קוד (אין צורך לכתוב תוכנית עובדת).

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

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

במקרה שאתם מניחים שהקוד עבור ביטויים בוליאניים הוא קוד "עם קפיצות" אז הממשק לפונקציה boolexp יהיה שונה. אפשרות אחת היא שהיא תקבל שני ארגומנטים: אחד הוא תווית אליה יש לקפוץ אם התנאי מתקיים. הארגומנט השני הוא תווית אליה יש לקפוץ אם התנאי אינו מתקיים. אפשר שאחד הארגומנטים יהיה FALL_THROUGH שמשמעותו: יש להמשיך לקוד שבא אחרי הקוד של הביטוי הבוליאני.

בכל מקרה אתם לא צריכים לכתוב את הפונקציה boolexp.

שאלה 4 (12%)

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 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. 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).

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.

בשני הסעיפים הבאים יש לכתוב סכימת תרגום.
ניתן להשתמש באותן פונקציות כמו בשאלה 2 (gen, newlabel, label). אין כאן צורך בפונקציה newtemp.

<u>סעיף א</u> (6%) הוסיפו פעולות סמנטיות ליצור קוד עבור מכונת מחסנית לכללי הגזירה הבאים:

```
expr -> expr ADDOP term | term
term -> term MULOP factor | factor
factor -> ( expr ) | ID | NUM
```

לאסימונים ADDOP ו- MULOP יש תכונה op המציינת את סוג האופרטור (פלוס או מינוס עבור ADDOP, כפל או חילוק עבור MULOP). ל- ID יש תכונה name ול- NUM יש תכונה val.

<u>סעיף ב</u> (6%) הוסיפו פעולות סמנטיות ליצור קוד עבור מכונת מחסנית לכלל הגזירה הבא:

stmt -> REPEAT (expr) stmt

המשמעות של המשפט היא: הביטוי expr מחושב פעם אחת. נסמן את התוצאה ב- n.

אז גוף הלולאה מבוצע n פעמים. אם n אינו מספר חיובי אז גוף הלולאה לא יבוצע אפילו פעם אחת.

בנוסף לפקודות של מכונת המחסנית שהוזכרו למעלה ניתן להשתמש גם בפקודות הבאות:
pop מוחקת את המספר שבראש המחסנית (עושה pop למחסנית)
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.

שאלה 5 (30%)

סעיף א

נגדיר שפה לכתיבת תוכניות פשוטות, המכילות הצהרות על משתנים ופקודות השמה.
לדוגמה:

```
1.  x , y: int;
2.  x := y ;
3.  begin   x  , w:  float    ;
4.          x := x + w  ;
5.  end
6.  y := x + y ;
7.  begin   y , w: int   ;
8.         x := w + y ;
9.          begin       a: int   ;
10.                     a := x ;
11.         end
12.end
```

המשתנים בשפה יכולים להיות מטיפוס int או float.
כל הצהרה מורכבת מרשימת משתנים ואחריהם נקודותיים וטיפוס.
בפקודות ההשמה מופיע באגף שמאל משתנה, ובאגף ימין ביטוי המורכב מסכום של מספר משתנים.

ניתן להגדיר בשפה זו בלוקים – כל בלוק מתחיל ב- begin ומסתיים ב- end. בתוך כל בלוק ניתן להגדיר משתנים, שהם מקומיים לאותו בלוק, וייתכן קינון של בלוקים. תחום ההשפעה של ההצהרות נקבע לפי "כלל הקינון הקרוב ביותר" (most closely nested rule), כפי שמוסבר בספר בעמ' 29-30 (סעיף 1.6.3) (וגם החל מעמוד 86 בסעיף 2.7.1).

טיפוסו של ביטוי:
אם הביטוי מכיל רק משתנים מטיפוס int, הטיפוס של הביטוי כולו הוא int.
אם קיים בביטוי משתנה אחד לפחות מטיפוס float, הטיפוס של הביטוי כולו הוא float.

המרות של טיפוסים:
כאשר האופרטור + מופעל על אופרנדים מטיפוסים שונים אז האופרנד שהוא מטיפוס int עובר המרה לטיפוס float לפני הפעלת האופרטור (והתוצאה אף היא מטיפוס float).

כאשר הטיפוס של הביטוי בצד ימין של משפט השמה שונה מהטיפוס של המשתנה בצד שמאל אז תוצאת הביטוי מומרת לטיפוס של המשתנה לפני ההשמה.

נגדיר דקדוק G אשר יוצר תוכניות בשפה המתוארת:

1. $P \rightarrow L \quad S$
2. $L \rightarrow L \quad D$
3. $L \rightarrow \varepsilon$
4. $D \rightarrow N \text{ ':' } T \text{ ';'}$
5. $N \rightarrow N \text{ ','} \quad ID$
6. $N \rightarrow ID$

```
7. T →  FLOAT
8. T →  INT
9. S →  S  C
10.S → ε
11.C →  id  ':='  E ';'
12.C →  BEGIN L  S END
13.E →  E  '+'  id
14.E →  ID
```

כתבו סכמת תרגום המבוססת על הדקדוק G, אשר מבצעת את הפעולות הבאות:
לכל הצהרה, המשתנים המוצהרים מוכנסים לטבלת הסמלים, יחד עם הטיפוס שלהם.

לפלט מודפס מספר ההמרות מטיפוס אחד לטיפוס אחר שהקוד מחייב.

הנחיות:

סעיפים 2.7.1 ו- 2.7.2 (החל מעמוד 86) עוסקים בטבלאות סמלים עבור scopes מקוננים.
הרעיון הבסיסי הוא להגדיר טבלת סמלים נפרדת עבור כל בלוק ולשמור את טבלאות הסמלים
במחסנית של טבלאות כאשר טבלת הסמלים של הבלוק הנוכחי תמיד בראש המחסנית.
הקוד בסכימות התרגום בספר הוא object oriented. ברור שזה לא מחויב
המציאות ואפשר לעשות דברים דומים גם בפסאודו קוד דמוי שפת C.

הגדירו פעולות על טבלת הסמלים לפי הצורך, למשל: `insert, lookup,`
`make_table, init`. תוכלו להשתמש בפעולות אלה בסכמת התרגום שלכם.
הסבירו בקצרה (בשורה אחת או שתיים) מה עושה כל אחת מהפעולות.

מותר להגדיר תכונות למשתני הדקדוק לפי הצורך.
הניחו שהתכונה `ID.name` מכילה את שמו של המשתנה ID

הגישה לטבלאות הסמלים יכולה להיעשות באמצעות משתנה (או משתנים) גלובליים.
המונה שסופר את מספר ההמרות גם הוא יכול להיות משתנה גלובלי. מעבר לכך אין לעשות
שימוש במשתנים גלובליים אלא בתכונות.

סעיף ב. ציירו את טבלאות הסמלים כפי שיראו בזמן שהקומפיילר נמצא בשורה 10 בדוגמא
שמופיעה למעלה.

שאלה 6 (10%)

סעיף א (5%). נתון מערך דו מימדי A:

```
double A[7][30]
```

נניח שגודל של double בזיכרון הוא 4 בתים (bytes).

נתון שהאיבר הראשון במערך A[0][0] נמצא בכתובת 100 בזכרון.
מה הכתובת של A[4][10] 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  foo {
    int p;
```

```
    struct foo* next;
};

int bar(struct foo **f[100],  double g);
```

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

שאלה 7 (12%)

סעיף א. (6%)

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

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 ו- free.

<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. (ברישום תוכן הרשימה – האוביקט השמאלי ביותר הוא בראש התור).

מי הם האוביקטים שיועברו לרשימה free ?

סעיף ב (6%)

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 2021**

מספר השאלות: 1

מועד משלוח המטלה: 19.2.2021

אנא שים לב:
מלא בדייקנות את הטופס המלווה לממ״ן
בהתאם לדוגמה המצויה בפתח חוברת המטלות
העתק את מספר הקורס ומספר המטלה הרשומים לעיל

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.

<u>שורת חותמת</u> – 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.

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 ישנם אסימונים הבאים:

Keywords

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

Symbols

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

Operators

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

More tokens

```
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
```

הבהרות:

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

המשמעות של static_cast דומה למשמעות בשפת C++.

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