ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – סתיו 2025א

כתבה: מיכל אבימור

אוקטובר 2024 – סמסטר סתיו – תשפ"ה

פנימי – לא להפצה.

תוכן העניינים

אל הסטודנט,

אני מקדמת את פניך בברכה, עם הצטרפותך אל הלומדים בקורס "מעבדה בתכנות מערכות". בחוברת זו תמצא את הדרישות לקבלת נקודות זכות בקורס, לוח הזמנים ומטלות הקורס.

לקורס קיים אתר באינטרנט בו תמצאו חומרי למידה נוספים, אותם מפרסם/מת מרכז/ת ההוראה. 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://telem.openu.ac.il

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

קורס זה הינו קורס מתוקשב. מידע על אופן ההשתתפות בתקשוב ישלח לכל סטודנט באופן אישי. ניתן להפנות שאלות בנושאי חומר הלימוד, והממ"נים לקבוצת הדיון של הקורס. בנוסף יופיעו שם הודעות ועדכונים מצוות הקורס. כניסה תכופה לאתר הקורס ולקבוצת הדיון שלה, מאפשרת לך להתעדכן בכל המידע, ההבהרות וכו' במסגרת הקורס.

ניתן לפנות אלי בשעות הייעוץ שלי (יפורסמו בהמשך באתר) או מחוץ לשעות הקבלה, באמצעות email, לכתובת: michav@openu.ac.il, ואשתדל לענות בהקדם.

- **שאילתא** - לפניות בנושאים אקדמיים שונים כגון מועדי בחינה מעבר לטווח זכאות ועוד, אנא עשו שימוש מסודר במערכת הפניות דרך שאילתא. לחצו על הכפתור פניה חדשה ואחר כך לימודים אקדמיים > משימות אקדמיות, ובשדה פניות סטודנטים: השלמת בחינות בקורס. המערכת תומכת גם בבקשות מנהלה שונות ומגוונות.

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.

אני מאחלת לך לימוד פורה ומהנה.

בברכה,

מיכל אבימור
מרכזת ההוראה בקורס.

1. לוח זמנים ופעילויות (קורס 20465 /א2025)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	01.11.2024-29.10.2024	ספר C פרקים 1-2-3	מפגש ראשון	
2	08.11.2024-03.11.2024	ספר C פרקים 1-2-3		
3	15.11.2024-10.11.2024	ספר C פרק 4	מפגש שני	
4	22.11.2024-17.11.2024	ספר C פרק 4		
5	29.11.2024-24.11.2024	ספר C פרק 5	מפגש שלישי	
6	06.12.2024-01.12.2024	ספר C פרק 5		ממ"ן 11 01.12.2024
7	13.12.2024-08.12.2024	ספר C פרק 6	מפגש רביעי	
8	20.12.2024-15.12.2024	ספר C פרק 6		

** 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	27.12.2024-22.12.2024 (ה-ו חנוכה)	ספר C פרק 6,7	מפגש חמישי	ממ"ן 12 22.12.2024
10	03.01.2025-29.12.2024 (א-ה חנוכה)	ספר C פרק 7		
11	10.01.2025-05.01.2025	ספר C פרק 7 + פרויקט	מפגש שישי	
12	17.01.2025-12.01.2025	פרויקט וחזרה		ממ"ן 22 12.01.2025
13	24.01.2025-19.01.2025	פרויקט וחזרה	מפגש שביעי	
14	31.01.2025-26.01.2025	פרויקט וחזרה	מפגש שמיני	ממ"ן 23 26.01.2025
15	03.02.2025-02.02.2025	פרויקט וחזרה		ממ"ן 14** 02.04.2025

מועדי בחינות הגמר יפורסמו בנפרד

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

***** לא תינתן דחייה בהגשת הפרויקט (ממ"ן 14), פרט למקרים חריגים של מילואים או אשפוז, במקרים אלו יש לתאם את מועד ההגשה מראש עם <u>מנחה</u> הקבוצה.***

2. תיאור המטלות

על מנת לתרגל את החומר הנלמד ולבדוק את ידיעותיך, עליך לפתור את המטלות המצויות בחוברת המטלות.

רוב המטלות בקורס זה הנן **מטלות חובה,** והן בעיקרן תוכניות מחשב. שתי מטלות הן רשות.
להלן מספרי המטלות ומשקליהן:

ממ"ן	משקל	פרקים
11	4 (ממ"ן חובה)	3,2,1
12	5 (ממ"ן חובה)	5,4
22	8 (ממ"ן רשות)	6,5,4
23	12 (ממ"ן רשות)	8,7,6
14	61 (ממ"ן חובה)	פרויקט גמר

עליך להגיש במהלך הקורס את כל מטלות החובה.
את התשובות לממ"נים יש להגיש באמצעות מערכת המטלות (במקרים מיוחדים ניתן להגיש את המטלות באמצעות הדואר או הגשה ישירה למנחה במפגשי ההנחיה. במקרה כזה יש לתאם את הדבר עם הבודק).
יש להגיש את קבצי המקור (.c, .h), קבצי ההרצה, קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (או צילומי מסך, אם לא נדרשו הקבצים הנ"ל).

הנחיות לכתיבת מטלות וניקודן

ניקוד המטלות ייעשה לפי המשקלים הבאים:

א. ריצה - 20%
התכנית עובדת על פי הדרישות בתרגיל, תוך השגת **כל** המטרות שהוגדרו. התכנית עוברת קומפילציה ללא הערות.

ב. תיעוד - 20%
התיעוד ייכתב בתוך הקוד. אין להוסיף הערות בקבצים נפרדים.
התיעוד יכלול:

1) הערה בראש תכנית שתכלול תיאור תמציתי של מטרות התכנית, כיצד מושגת מטרה זו, תיאור המודלים והאלגוריתם, קלט/פלט **וכל הנחה** שהנכם מניחים.
2) לכל מציג (אב-טיפוס) prototype של פונקציה (בקובץ ה-header הצמוד לקוד), יוצמד תיאור של קלט/פלט, ופעולת הפונקציה. **מטרה:** זהו קובץ היצוא ועל כן עליו להסביר למי שאין לו גישה לקוד איך עליו להשתמש בפונקציה.
3) לפני הכותרת (header) של כל פונקציה יבוא תיאור של פעולתה, הנחות ואלגוריתם.

מטרה: התיעוד לפני כל פונקציה נועד לתת היכרות ראשונית, לפעולת הפונקציה, תוך פירוט כיצד הפונקציה עושה זאת. תיעוד זה אמור לאפשר לקורא את הקוד (שלא כתב את הקוד), להבין את הקוד.

4) לכל משתנה יהיה שם משמעותי ויוצמד אליו תיעוד לגבי תפקודו בתכנית. i,j,k - משמשים בד"כ כשמות אינדקסים ואין צורך לתעד אותם.

5) לא יופיעו "מספרי קסם" בגוף התכנית למעט 0,1 לאיתחול משתני לולאות. יש להשתמש בקבועים בעלי שמות משמעותיים שיכתבו באותיות גדולות, ויתועדו בשלב ההגדרה. כל טיפוס מורכב יוגדר כ- typedef ויתועד. נהוג לקרוא לטיפוסים מורכבים בשמות משמעותיים ולהשתמש באותיות גדולות.

6) יש להשתמש בשמות משמעותיים ל: פונקציות, מקרואים, משתנים, קבועים, הגדרת טיפוסים וקבצים.

7) יש להקפיד על קריאות ובהירות תוך שימוש באינדנטציה (היסח) מסודרת ואחידה.

ג. תכנות - 40%

יש להקפיד על כתיבה מסודרת ומודולרית של קוד:

- חלוקה לקבצים - כשלכל קובץ מוצמד קובץ header אם צריך (כאשר נדרש בתרגיל).
- חלוקה לפונקציות.
- שימוש במקרואים.
- שימוש נכון ב-MAKEFILE, (במיוחד כאשר אתם נדרשים לחלק את התוכנית למספר קבצים, במסגרת הממ"ן).
- הסתרת אינפורמציה - ושימוש בהפשטת מידע.
- הימנעות ככל שניתן משימוש במשתנים גלובליים.
- שימוש מירבי ונכון במלוא הכלים שמעמידה השפה לרשותכם.
- קוד אלגנטי ולא מסורבל.

ד. יעילות התכנית והתרשמות כללית - 20%

המשקלים הנ"ל מהווים קו מנחה לחלוקת הנקודות. מובן שתהיה התייחסות לכל תכנית לגופה, בהתאם למידת המורכבות של התרגיל.

ינתנו קנסות במיקרים הבאים:

- אי הגשת קבצי סביבה – MAKEFILE – 20 נקודות.
- עבור אותם ממ"נים בהם מוגדר שם קובץ, פונקציה, או פרמטר, שימוש בשם שונה מזה המוגדר בממ"ן – 10 נקודות.

להשומת לבך: חל איסור מוחלט של הכנה משותפת של מטלות או העתקת מטלות. תלמיד שייתפס באחד מאיסורים אלה ייענש בהתאם לנאמר בתקנון המשמעת נספח 1 בידיעון של האו"פ. רק את ממ"ן 14 מותר להגשה בזוגות (**לא** ניתן להגיש בשלשות!), כאשר שני הסטודנטים המגישים שיכים לאותה קבוצת לימוד.

3. התנאים לקבלת נקודות זכות בקורס

א. להגיש את מטלות החובה בקורס (11, 12) וכן את פרויקט הגמר (14).

ב. ציון של לפחות 60 נקודות בבחינת הגמר ובפרויקט הגמר.

ג. ציון סופי בקורס של 60 נקודות לפחות.

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 והפרויקט בציון 60 ומעלה והגישו מטלות כנדרש באותו קורס.

מטלת מנחה (ממ"ן) 11

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 1,2,3

מספר השאלות: 2 **משקל המטלה:** 4 נקודות (**חובה**)

סמסטר: 2025א' **מועד אחרון להגשה:** 01.12.2024

אופן ההגשת המטלות:
שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic . יש להגיש את קבצי המקור c. (h. רק אם קיימים), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר).
הקבצים של כל תוכנית יהיו בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.
יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה, ומאפשרות בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.
את המטלה יש להגיש בקובץ zip.

חשוב מאוד:

- **אופן הגשת המטלה והקבצים** הנדרשים להגשה מופיעים כאן וכן בעמודים ה-ז בסעיף תיאור המטלות. במקרה של הנחייה שונה בפורום, יש לוודא את הנושא עם מנחה הקבוצה.
- לאחר ההגשה, יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן **תקין**.

שאלה 1 (תכנית ראשית בקובץ letters. c) (50 נקודות)

עליכם לכתוב תכנית הקולטת מהקלט הסטנדרטי טקסט (רצף בקוד אסקי), עד EOF (סוף הקלט), ומדפיסה את הקלט לפלט הסטנדרטי בשינויים הבאים:

1. בתחילת כל משפט, אם התו הראשון הוא אות אלפבתית קטנה, יש להמירה לאות גדולה.
2. בכל טקסט בין מרכאות כפולות יש להמיר כל אות אלפבתית קטנה לאות גדולה.
3. בכל מקום אחר בטקסט (שלא לפי סעיפים 2-1 לעיל), יש להמיר כל אות אלפבתית גדולה לאות קטנה.
4. ספרות (התוים '9'–'0') לא יודפסו לפלט, ויש לדלג עליהן (לא יודפס דבר במקומן).
5. כל תו (קוד אסקי) שאינו אות אלפבתית או ספרה יודפס ללא שינוי, לרבות סימני פיסוק וכל התווים הלבנים (רווח, טאב, שורה-חדשה) בכל מקום בטקסט

דרישות נוספות:

- משפט מסתיים בתו נקודה ('.') בלבד.
- משפט חדש מתחיל בתו הראשון שאינו לבן אחרי סוף המשפט הקודם.
- התו נקודה המופיע בטקסט בין מרכאות כפולות אינו נחשב כסוף משפט.
- משפט יכול להתפרס על יותר משורה אחת בקלט.
- טקסט בין מרכאות כפולות יכול להתפרס על יותר משורה אחת בקלט.
- מותרות שורות ריקות בקלט (יודפסו לפלט כשורות ריקות).
- מותרים משפטים "ריקים", המכילים מכילים רק נקודה. יש להדפיס גם משפט ריק.
- מספר השורות בקלט בלתי מוגבל, ואורך כל שורת קלט בלתי מוגבל.
- הקלט מסתיים כשהתכנית מזהה בקלט מצב EOF. אפשר לדמות מצב EOF במקלדת באמצעות הקלדה של צרוף המקשים ctrl+d באובונטו, או ctrl+z בחלונות
- הקלט יכול להסתיים (EOF) גם באמצע משפט, כלומר ללא נקודה, ואף ללא סגירת מרכאות.

לדוגמה, עבור הקלט הבא (9 שורות):

```
I am young. You are young. All of us are young.
"I think we  need   some help. Please" HELP. NO,    NO   NO,
I DO NOT
    NEED HELP

WHATSOEVER.
"Today's date is
    15/2/2021"...
I am 18  years old, are you 20 years old? Maybe    30 years?
```

יודפס הפלט הבא (9 שורות):

```
I am young. You are young. All of us are young.
"I THINK WE  NEED   SOME HELP. PLEASE" help. No,    no   no,
i do not
    need help

whatsoever.
"TODAY'S DATE IS
    //"...
I am   years old, are you  years old? maybe     years?
```

על התוכנית **להדפיס הודעת בקשה ידידותית לקלט** המפרטת מה על המשתמש להקליד.
הניחו כי הקלט תקין, כלומר אין צורך לבדוק שגיאות בקלט.

הקלט לתוכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התוכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית.

חובה לצרף להגשה הרצות בדיקה (אחת או יותר), המדגימות את פעולת התוכנית על קלט מגוון.
יש להדגים את כל השינויים 1-5 המתוארים לעיל , תוך עמידה בכל הדרישות הנוספות לעיל.
יש להגיש תדפיסי מסך של כל ההרצות וכן את קבצי הקלט.

שאלה 2 (תכנית ראשית בקובץ xy_bits) (50 נקודות)

עליכם לכתוב תכנית המקבלת כקלט משתנים x, y מסוג unsigned long. מטרת התכנית, עבור המשתנה x: להדליק את הביט במקום ה-13 מימין (ביט מספר 12). עבור המשתנה y: לבדוק האם דלוק הביט במקום ה-7 מימין (ביט מספר 6). <u>הערה</u>: מספור הביטים מתחיל מ 0 .

יש להדפיס את הקלט מיד עם קליטתו ולבסוף להדפיס את התוצאות בצורה ברורה ומובנת. לדוגמה, אם:

למשתנה x הוכנס ערך המיוצג בביטים 00101000111110010010100011111001

למשתנה y הוכנס ערך המיוצג בביטים 11000011001110001100001100111000

בסיום על התכנית להדפיס, את השינוי עבור משתנה x - 00101000111110010011100011111001,

ועבור משתנה y - האם המקום ה-7 דלוק – ביט מספר 6 (YES/NO): בדוגמה שלנו NO

<u>הנחיות והערות נוספות</u>:

- יש להשתמש בפונקציית / פונקציות עזר לטיפול בביטים
- בתחילת הריצה, על התוכנית **להדפיס הודעת בקשה ידידותית לקלט,** המפרטת מה על המשתמש להקליד.
- הניחו כי הקלט תקין, כלומר אין צורך לבדוק שגיאות בקלט.
- על התכנית להדפיס באופן נאה **בבסיס 2** את האופרנדים ואת התוצאות הסופיות. כמו כן, התכנית תדפיס זאת גם כמספרים מטיפוס **unsigned long**.

חובה לצרף להגשה מספר הרצות בדיקה (לפחות שתי הרצות), המדגימות את פעולת התכנית על מגוון נתוני קלט. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות**. במידה ותשתמשו בקבצי קלט, **יש להגיש גם קבצים אלה**.

<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ"ן, פרט למקרים חריגים כגון אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש ממנחה הקבוצה**.

בהצלחה!

מטלת מנחה (ממ"ן) 12

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5

מספר השאלות: 1 **משקל המטלה:** 5 נקודות **(חובה)**

סמסטר: 2025א' **מועד אחרון להגשה:** 22.12.2024

אופן הגשת המטלות: שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס **הסבר מפורט ב"נוהל הגשת מטלות מנחה"**

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic . יש להגיש את קבצי המקור c. (h. אם קיימים), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר).
קבצי התוכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.
יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.
את המטלה יש להגיש בקובץ zip.

לאחר ההגשה, יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן **תקין**.

שאלה 1 (תכנית ראשית בקובץ adjacency.c)

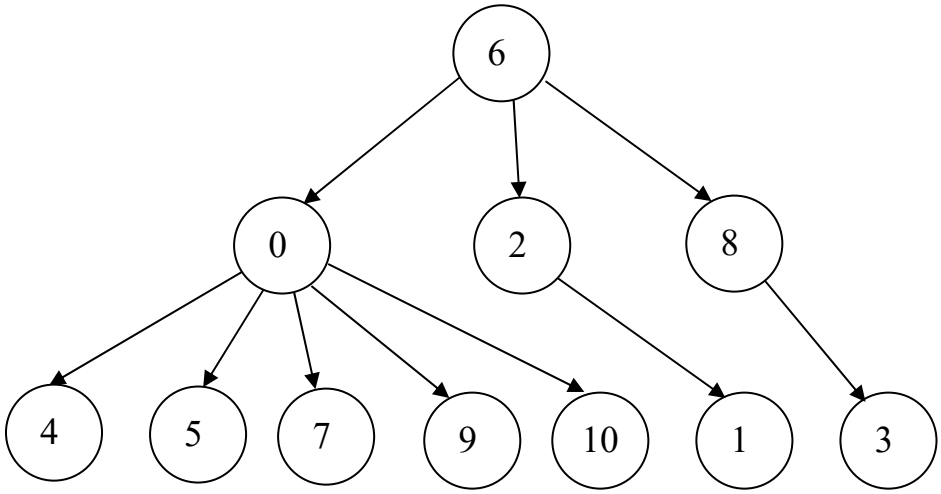
נתון עץ משרש מכוון T בעל N צמתים. "שמות" הצמתים הם האינדקסים מ-0 ועד N–1.

העץ מיוצג על ידי מטריצה A בגודל NxN באופן הבא:

A[u][v] == 1 אם קיימת קשת מכוונת מהצומת u לצומת v בעץ T, או במילים אחרות, אם u הוא האב של v בעץ. אחרת, A[u][v]==0.

המטריצה A נקראת **מטריצת השכנויות** של העץ.

בדוגמה הבאה מוצג עץ בעל N=11 צמתים.



עץ זה מיוצג על ידי מטריצת השכנויות A הבאה
(השורה העליונה והעמודה השמאלית הם האינדקסים של איברי המטריצה):

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	1	1	0	1	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	1	0	1	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0

עליכם לכתוב תוכנית לפי הדרישות שלהלן.

א. הגדירו בעזרת #define ו/או enum את N, ושני קבועים TRUE ו-FALSE.

ב. הגדירו בעזרת typedef טיפוס בשם adjmat אשר מחזיק מטריצת שכנויות בגודל NxN. שימו לב שממדי המטריצה תלויים בקבוע N שהגדרתם.

ג. כתבו פונקציה בשם path המקבלת שלשה פרמטרים: מטריצת שכנויות מטיפוס adjmat המייצגת עץ מושרש מכוון, וכן שני אינדקסים של צמתים u ו-v. הפונקציה מחזירה את הערך TRUE אם קיים מסלול מכוון (לפי כיווני החיצים) מהצומת u אל הצומת v בעץ המיוצג על ידי המטריצה, ואחרת מחזירה FALSE. במילים אחרות, מוחזר הערך TRUE אם u הוא אב-קדמון או אב ישיר של v בעץ, ואחרת מוחזר FALSE.

עבור הדוגמה לעיל של המטריצה A, הקריאה path(A,6,1) תחזיר TRUE, וכך גם path(A,0,9).
לעומת זאת, הקריאה path(A,2,10) תחזיר FALSE, וכך גם path(A,1,2).

אם אחד האינדקסים u או v חורג מממדי המטריצה, הפונקציה תחזיר FALSE.
אם שני האינדקסים זהים ואינם חורגים מהמטריצה, הפונקציה תחזיר TRUE.

<u>הערה</u>: הקבוע N אמור להיות נגיש בכל חלקי התוכנית, ואין צורך להעבירו כפרמטר לפונקציה.

ד. כתבו תכנית ראשית (הפונקציה main), המבצעת כדלקמן.

1) התוכנית תגדיר משתנה מהטיפוס adjmat (כלומר מופע של מטריצת שכנויות בגודל NxN).

2) התוכנית תבקש מהמשתמש רשימת ערכים עבור אברי המטריצה (הערכים 0 או 1). לאחר קליטת כל נתוני המטריצה והצבתם במשתנה, התוכנית תדפיס את המטריצה בתצוגה דו-ממדית נאה.

<u>לתשומת לב</u>:

- על התוכנית לעבוד נכון עבור מטריצה <u>בכל גודל</u>, תוך שינוי הגדרת הקבוע N בלבד (וכמובן קימפול מחדש). <u>הניחו</u> שהמשתמש <u>אינו יודע מראש</u> מהם ממדי המטריצה בהרצה הנוכחית, ולפיכך יש לדווח לו את הערך N באמצעות הודעת בקשה לקלט.
- תוכלו לארגן בקלט את נתוני המטריצה בכל דרך הנוחה לכם. למשל, אפשר להעביר את כל איברי המטריצה בשורת קלט בודדת, לפי סדר השורות במטריצה. אפשרות אחרת היא להעביר בכל שורת קלט שורה אחת של המטריצה. אפשרות נוספת היא להעביר כל איבר בשורת קלט נפרדת.

3) אחרי הדפסת מטריצת השכנויות, התוכנית תבקש מהמשתמש שני אינדקסים של צמתים, ותקרא לפונקציה path עם המטריצה וזוג האינדקסים. אחרי החזרה מהפונקציה, התוכנית תדפיס הודעה נאה הכוללת את זוג האינדקסים ואת התוצאה שהוחזרה.

4) לאחר מכן, התוכנית תבקש מהמשתמש זוג אינדקסים נוסף, ותפעל עליהם באותו אופן (כמפורט לעיל בסעיף ד3). התוכנית תמשיך לקלוט ולטפל בזוגות של אינדקסים בזה אחר זה, ותסתיים כאשר יועבר בקלט הזוג $-1,-1$ (או כאשר יתגלה בקלט מצב של EOF).

<u>לתשומת לב</u>: אין לצאת מהתוכנית על ידי "הריגה" (למשל באמצעות הקלדת ctrl-c).

אפשר להשתמש בפונקציות עזר נוספות (למשל, פונקציה להדפסת המטריצה).

הניחו שהקלט תקין (למעט אינדקסים חורגים, כאמור לעיל בסעיף ג'). אין צורך לטפל בשגיאות בקלט.

הקלט לתוכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התוכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית.

על התוכנית **להדפיס הודעת בקשה ידידותית <u>בכל פעם</u> כשנדרש קלט** (שימו לב גם לסעיף ד2 לעיל).

חובה לצרף להגשה מספר <u>הרצות בדיקה</u> המדגימות את פעולת התוכנית על עצים בגדלים שונים ומגוון מסלולים בכל עץ. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל הרצות הדוגמה**. במידה ותשתמשו בקבצי קלט, **יש להגיש גם קבצים אלה**.

<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ"ן, פרט למקרים חריגים כגון אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

בהצלחה!

מטלת מנחה (ממ"ן) 22

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5,6

מספר השאלות: 1 **משקל המטלה:** 8 נקודות (רשות)

סמסטר: 2025א' **מועד אחרון להגשה:** 12.01.2025

אופן הגשת המטלות: שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס **הסבר מפורט ב"נוהל הגשת מטלות מנחה"**

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic . יש להגיש את קבצי המקור c. (h. אם קיימים), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר).

קבצי התוכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.

את המטלה יש להגיש בקובץ zip.

חשוב מאוד:

- **אופן הגשת המטלה והקבצים** הנדרשים להגשה מופיעים כאן וכן בעמודים ה-ז בסעיף תיאור המטלות. במקרה של הנחייה שונה בפורום, יש לוודא את הנושא עם מנחה הקבוצה.
- לאחר ההגשה, יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן **תקין**.

שאלה 1 (בקבצים עיקריים mycomp.c, complex.c, complex.h)

עליכם לכתוב תוכנית מחשב אינטראקטיבית הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן. הפקודות עוסקות בפעולות אריתמטיות על מספרים מרוכבים.

תזכורת מספרים מרוכבים:

מספר מרוכב (complex) הוא מספר בן שני חלקים: חלק ממשי וחלק מדומה, כאשר ביניהם רשום הסימן "+" או הסימן "–".

מבנה המספר הוא: a + bi כאשר a החלק הממשי ו- bi החלק המדומה. החלק המדומה הוא מכפלה של שני גורמים: b הוא מספר ממשי, ואילו i מציין את השורש הריבועי של המספר 1-, כלומר $i = \sqrt{-1}$.

דוגמאות של מספרים מרוכבים:

24.65 + (15.376)i 87.5 – (14.3)i –153+24i

<u>להלן הגדרות של הפעולות החשבוניות הבסיסיות על מספרים מרוכבים:</u>

חיבור של שני מספרים מרוכבים:

(a + bi) + (c + di) = (a + c) + (b + d)i

חיסור של שני מספרים מרוכבים:

(a + bi) – (c + di) = (a – c) + (b – d)i

כפל של מספר מרוכב במספר ממשי:

m∗ (a + bi) =ma + (mb)i

כפל של מספר מרוכב במספר מדומה:

mi∗ (a + bi) = mi∗ a + mi∗ bi = –mb + (ma)i

כפל של מספר מרוכב במספר מרוכב:

(a + bi)∗ (c + di) = a∗ c + a∗ di + bi∗ c + bi∗ di = (ac – bd) + (ad + bc)i

חישוב הערך המוחלט של מספר מרוכב (התוצאה היא מספר ממשי אי-שלילי):

$$|a + bi| = \sqrt[2]{a^2 + b^2}$$

<u>משימות התוכנית:</u>

עליכם להגדיר, באמצעות שימוש ב- typedef את הטיפוס complex אשר מחזיק מספר מרוכב.
על מבנה הנתונים שבחרתם להיות יעיל מבחינת כמות זיכרון הנדרשת, ויעיל מבחינת הגישה אליו.

בנוסף, עליכם להגדיר בתוכנית הראשית 6 משתנים: A,B,C,D,E,F מטיפוס complex.
בתחילת הריצה, יש לאתחל את כל ששת המשתנים לאפס (הערך המרוכב 0+0i).

כעת, עליכם לבצע פעולות חשבוניות עם מספרים מרוכבים. כל פעולה תופעל באמצעות פקודה שמועברת בקלט לתוכנית, כמפורט להלן. בפקודות אלה, אופרנד שהוא משתנה מרוכב יהיה אחד משש המשתנים שהוגדרו לעיל.

<u>מפרט הפקודות המשמשות כקלטים לתוכנית:</u>

1. הצבת מספר מרוכב במשתנה:

מספר-ממשי-שני ,מספר-ממשי-ראשון ,שם-משתנה-מרוכב read_comp

הפקודה תגרום להצבת ערך מרוכב במשתנה המרוכב ששמו מופיע בפקודה. המספר הממשי הראשון הוא החלק הממשי של המספר המרוכב, והמספר הממשי השני הוא החלק המדומה של המספר המרוכב (החלק המדומה נתון בפקודה ללא הגורם i)

לדוגמה, הפקודה הבאה:

read_comp A, 5.1, 6.23

תבצע את ההצבה:

A = 5.1 + (6.23)i

<u>הערה</u>: זוהי הפקודה היחידה שמשנה את ערכו של משתנה מרוכב בתוכנית.

2. הדפסת ערך של משתנה מרוכב:

שם-משתנה-מרוכב print_comp

ערכו של המשתנה המרוכב ששמו ניתן בפקודה יודפס בצורה נאה בפלט.

לדוגמה, הפקודה הבאה:

print_comp A

תגרום להדפסת ערך המשתנה A. בהנחה שהפקודה היא בהמשך לדוגמה בסעיף 1, יודפס:

5.10 + (6.23)i

הערה: יש להדפיס כל מספר עם דיוק של לפחות שתי ספרות מימין לנקודה.

3. חיבור מספרים מרוכבים:

שם-משתנה-מרוכב-ב' ,שם-משתנה-מרוכב-א' add_comp

יתבצע חיבור של שני המספרים המרוכבים אשר במשתנים המופיעים בפקודה:

מספר-מרוכב-ב' + מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

4. חיסור מספרים מרוכבים:

שם-משתנה-מרוכב-ב' ,שם-משתנה-מרוכב-א' sub_comp

יתבצע חיסור של המספר המרוכב במשתנה ב׳ מן המספר המרוכב במשתנה א׳:

מספר-מרוכב-ב׳ – מספר-מרוכב-א׳

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

5. כפל מספר מרוכב עם מספר ממשי:

מספר-ממשי ,שם-משתנה-מרוכב mult_comp_real

יתבצע כפל של המשתנה המרוכב והמספר הממשי הנתונים בפקודה.

מספר-ממשי * מספר-מרוכב

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

6. כפל מספר מרוכב עם מספר מדומה:

מספר-מדומה ,שם-משתנה-מרוכב mult_comp_img

יתבצע כפל של המשתנה המרוכב והמספר המדומה הנתונים בפקודה.

המספר המדומה נתון בפקודה ללא הגורם i.

(i * מספר-מדומה) * מספר-מרוכב

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

7. כפל שני מספרים מרוכבים:

שם-משתנה-מרוכב-ב׳ ,שם-משתנה-מרוכב-א׳ mult_comp_comp

יתבצע כפל של שני המשתנים המרוכבים המופיעים בפקודה:

מספר-מרוכב-ב׳ * מספר-מרוכב-א׳

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

8. ערך מוחלט של מספר מרוכב:

שם-משתנה-מרוכב abs_comp

יחושב ערכו המוחלט של המשתנה המרוכב שמופיע בפקודה:

| מספר-מרוכב |

תוצאת הפעולה תודפס לפלט בפורמט דומה לפורמט ההדפסה של סעיף 2.

9. סיום התוכנית:

stop

פקודה זו היא ללא פרמטרים, ומטרתה לסיים את התוכנית.

המבנה התחבירי של הקלט:

- כל פקודה תופיע בשלמותה בשורת קלט יחידה, כולל כל הפרמטרים. מותרות גם שורות ריקות (שורות המכילות רק תווים לבנים).
- שם הפקודה מופרד מהפרמטר הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).
- בין כל שני אופרנדים יש פסיק אחד. לפני ואחרי הפסיק יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת. אסור שיהיה פסיק אחרי הפרמטר האחרון.
- יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת לפני שם הפקודה, וגם בסוף השורה (אחרי הפרמטר האחרון).
- אסור שיהיו תווי זבל בסוף השורה (למעט תווים לבנים).
- שמות הפקודות יופיעו באותיות קטנות בלבד, ושמות המשתנים באותיות גדולות בלבד.

אופן פעולת התוכנית:

יש לממש ממשק משתמש ידידותי, כך שהמשתמש יוכל להבין בכל שלב של התוכנית מה עליו לעשות. בפרט, על התוכנית להודיע באמצעות הודעה או סימן (prompt) בכל פעם שהיא מוכנה לקלוט את הפקודה הבאה. התוכנית תמשיך לקלוט ולבצע פקודה אחרי פקודה, עד שתתקבל הפקודה stop.

התוכנית אינה מניחה שהקלט תקין. על התוכנית לנתח כל פקודה ולוודא שאין בה שגיאות (ראו דוגמאות בהמשך). במידה ונתגלתה שגיאה, התוכנית תדפיס הודעת שגיאה פרטנית, ותמשיך לפקודה הבאה, בלי לבצע את הפקודה השגויה. אין לעצור את התוכנית עם גילוי השגיאה הראשונה. אין צורך לדווח על יותר משגיאה אחת בכל שורת קלט.

יש לטפל גם במצב של EOF (גמר הקלט). עצירת התוכנית שלא באמצעות פקודת stop אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ באמצעות redirection), ויש להדפיס הודעת שגיאה על כך ורק אז לעצור. שימו לב: השורה האחרונה בקובץ קלט אינה חייבת להסתיים בתו '\n'. במקרה כזה, אם יש בשורה האחרונה פקודה, יש לטפל בה כרגיל (סוף הקובץ מסמן את סוף הפקודה).

להלן דוגמאות של קלט שגוי:

שימו לב: ייתכנו סוגים נוספים של שגיאות בקלט.
עליכם לחשוב על כל מגוון השגיאות האפשריות, ולטפל בכולן.

1. לפקודה:

read_comp G, 3.1, 6.5

יש להגיב בהודעה כגון:

Undefined complex variable

2. לפקודה:

read_comp a, 3.6, 5.1

יש להגיב בהודעה כגון:

Undefined complex variable

3. לפקודה:

do_it A, B

יש להגיב בהודעה כגון:

Undefined command name

4. לפקודה:

Add_Comp A, C

יש להגיב בהודעה כגון:

Undefined command name

5. לפקודה:

read_comp A, 3.5, xyz

יש להגיב בהודעה כגון:

Invalid parameter – not a number

6. לפקודה:

read_comp A, 3.5

יש להגיב בהודעה כגון:

Missing parameter

7. לפקודה:

read_comp A, 3.5, -3,

יש להגיב בהודעה כגון:

Extraneous text after end of command

8. לפקודה:

add_comp B

יש להגיב בהודעה כגון:

Missing parameter

9. לפקודה:

print_comp C, D

יש להגיב בהודעה כגון:

Extraneous text after end of command

10. לפקודה:

sub_comp F, , D

יש להגיב בהודעה כגון:

Multiple consecutive commas

11. לפקודה:

mult_comp_comp F D

יש להגיב בהודעה כגון:

Missing comma

12. לפקודה:

mult_comp_real, A, 2.5

יש להגיב בהודעה כגון:

Illegal comma

13. לפקודה:

mult_comp_img A, B

יש להגיב בהודעה כגון:

Invalid parameter – not a number

14. לפקודה:

abs_comp

יש להגיב בהודעה כגון:

Missing parameter

15. לפקודה:

abs_comp 2.5

יש להגיב בהודעה כגון:

Undefined complex variable

16. לפקודה:

stop A

יש להגיב בהודעה כגון:

Extraneous text after end of command

<u>להלן דוגמה של סדרת פקודות שכולן תקינות:</u>

הערה: סדרה כגון זו יכולה לשמש כקלט בהרצת בדיקה של התוכנית על קלט תקין.

```
print_comp A
print_comp B
print_comp C
read_comp A, 45.1, -23.75
print_comp A
read_comp B, 54.2, 3.56
print_comp B
read_comp C, 0, -1
```

```
print_comp C
add_comp A, B
sub_comp C, A
sub_comp B, B
sub_comp D, A
mult_comp_real A, 2.51
mult_comp_img A, -2.564
mult_comp_comp A, B
mult_comp_comp   E  ,   C
abs_comp A
abs_comp B
abs_comp C
abs_comp F
stop
```

<u>דרישות והנחיות נוספות</u>:

- יש לחלק את התוכנית למספר קבצי מקור: mycomp.c, complex.c, ו-complex.h.
  - בקובץ mycomp.c תהיה התוכנית הראשית main, וכן כל פעילויות האינטראקציה עם המשתמש וניתוח הקלט (לרבות הדפסת הודעות השגיאה). כמו כן, יוגדרו בקובץ זה ששת המשתנים מטיפוס complex.
  - בקובץ complex.c יש לרכז את הפעולות החשבוניות על מספרים מרוכבים. לכל פעולה יש לממש פונקציה נפרדת, עם פרמטרים לפי מפרט הפעולה המוגדר לעיל. אין לבצע ניתוח של הקלט או הדפסות מתוך קובץ זה, למעט הדפסת המספר המרוכב כנדרש בפעולה print_comp.
  - בקובץ complex.h תהיה הגדרת טיפוס הנתונים complex, וכן ההצהרות (אב-טיפוס) של הפונקציות הממומשות בקובץ complex.c. יש לכלול (#include) את הקובץ complex.h בקבצי המקור האחרים.
  - באפשרותכם לבנות קבצי מקור נוספים (למשל: קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').
- הקלט לתוכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התוכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית. בכל קובץ קלט תהיה סדרה של פקודות על מספרים מרוכבים.
- לפני הניתוח של כל שורת קלט, על התוכנית **להדפיס באופן יזום את השורה לפלט, בדיוק כפי שנקראה**. זאת כדי שניתן יהיה לראות בפלט את הפקודות המקוריות, גם כאשר הקלט מגיע מקובץ.

חובה לצרף להגשה הרצות בדיקה (אחת או יותר), המדגימות את השימוש בכל סוגי הפקודות ובכל ששת המשתנים המרוכבים, וכן את הטיפול בכל מגוון השגיאות בקלט.
<u>רמז</u>: מומלץ להכניס בקלט פקודת הדפסה של התוצאה אחרי כל פעולה, כדי להראות שהתוצאה אכן נכונה (ראו לעיל הדוגמה של סדרת פקודות תקינות).
יש להגיש תדפיס מסך (וקובץ פלט) של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ"ן, פרט למקרים חריגים כגון אשפוז. במקרים אלו יש לבקש ולקבל אישור מראש ממנחה הקבוצה.

בהצלחה!

מטלת מנחה (ממ"ן) 23

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 6,7

מספר השאלות: 2 **משקל המטלה**: 12 נקודות (רשות)

סמסטר: 2025א' **מועד אחרון להגשה:** 26.01.2025

אופן הגשת המטלות:
שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic . יש להגיש את קבצי המקור c. (h. נדרשים במטלה זו), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר).קבצי התכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c. יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית. את המטלה יש להגיש בקובץ zip.

<u>חשוב מאוד</u>:

- **אופן הגשת המטלה והקבצים** הנדרשים להגשה מופיעים כאן וכן בעמודים ה-ז בסעיף תיאור המטלות. במקרה של הנחייה שונה בפורום, יש לוודא את הנושא עם מנחה הקבוצה.
- לאחר ההגשה, יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן **תקין**.

שאלה 1 (10 נקודות)

בכל סעיף, עליכם לכתוב האם "**תמיד נכון**" בשפת ANSI-C, "**לפעמים נכון ולפעמים אינו נכון**", או "**תמיד אינו נכון**". עליכם לנמק את תשובתכם. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות (כל סעיף 5 נקודות).

א. ההחלטה איזה סוג של חילוק יתבצע, חילוק בין מספרים ממשיים או בין מספרים שלמים, מתבצעת על-פי ההקשר (context).

ב. העברת רשומה - structure - לפונקציה, אפשרית רק לפי כתובת.

<u>לתשומת לב</u>:

את הפתרון לשאלה זו יש להגיש במסמך (קובץ) <u>מוקלד</u>, בפורמט <u>word</u> או pdf.

שאלה 2 (90 נקודות) (תכנית ראשית בקובץ timediff.c)

(10 נקודות) א. עליכם להגדיר טיפוס נתונים בשם time המשמש לייצוג זמן. המידע המאוחסן הוא:

<u>תאריך (לפי הלוח הגרגוריאני)</u>: שנה, חודש, יום.

<u>שעת היום</u>: שעה (לפי 24 שעות), דקה, שניה.

(80 נקודות) ב. על התכנית הראשית (הפונקציה main), לקבל שם של קובץ טקסט **כארגומנט של שורת פקודה**. יש לבנות קובץ המכיל מספר שורות:
כל שורה בקובץ תכיל 10 מספרים: 5 הראשונים עבור התאריך הראשון ו- 5 הבאים עבור התאריך השני, על פי הסדר המופיע בסעיף א.

עליכם לכתוב פונקציה בשם time_diff המקבלת שני פרמטרים מטיפוס time, ומחזירה את ההפרש בין שני הזמנים ביחידות של שניות. על הערך המוחזר להיות תמיד חיובי, אם כי <u>לא ניתן</u> להניח שיש סדר קבוע בין הזמנים של שני הפרמטרים. כלומר, על הפונקציה לבדוק מי משני הפרמטרים הוא הזמן המוקדם יותר ומי המאוחר יותר.

התכנית ראשית (הפונקציה main), תקרא בכל פעם שורה מהקובץ, תאתחל שני משתנים מטיפוס time (על פי השורה שנקראה) ותקרא לפונקציה time_diff. התכנית תדפיס לפלט בפורמט נאה ומפורט את שני הזמנים (תאריך ושעת היום), ואת ההפרש ביניהם (בשניות).

באופן זה הפונקציה time_diff תופעל מספר פעמים <u>באותה הרצה</u>, עד לסיום הקובץ, עם שורות של ערכים מגוונים של זמנים, כדי להדגים את המקרים השונים של עבודת הפונקציה.

הנחיות נוספות:

- יש לטפל בשגיאות: למשל: מקרה הקובץ לא קיים / לא נפתח. ארגומנט לא קיים / לא נכון
- לשם פשטות, מותר להניח כי בחודש פברואר יש תמיד 28 ימים.
- שנים שלפני הספירה יצוינו באמצעות מספרים שליליים.
- ניתן להניח שכל ערך שיחושב לא יגלוש מהתחום המוגדר של הטיפוס.
- אין להשתמש בפונקציות של הספרייה הסטנדרטית המטפלות בזמן, כגון time או difftime.
- אין לבצע קלט נתוני תאריכים מהמשתמש. כל **הנתונים** יוכנו בקובץ, ויהיו **תמיד תקינים**.

חובה לצרף להגשה <u>הרצת בדיקה</u> המדגימה את פעולת התכנית על נתונים מגוונים. ואת קובץ הקלט.
עבור ההגשה שם הקובץ יהיה **timediff.txt**. יש להגיש **תדפיס מסך (או קובץ פלט)** של ההרצה.
<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ"ן, פרט למקרים חריגים כגון אשפוז. במקרים אלו יש לבקש ולקבל אישור מראש ממנחה הקבוצה.

בהצלחה!

מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 | משקל המטלה: 61 נקודות **(חובה)**

סמסטר: 2025א' | מועד אחרון להגשה: 02.04.2025

קיימת אפשרות אחת להגשת המטלה:
שליחה באמצעות מערכת המטלות המקוונת באתר הבית של הקורס

הסבר מפורט ב"נוהל הגשת מטלות מנחה"

אחת המטרות העיקריות של הקורס "20465 - מעבדה בתכנות מערכות" היא לאפשר ללומדים בקורס להתנסות בכתיבת פרויקט תוכנה גדול, אשר יחקה את פעולתה של אחת מתוכניות המערכת השכיחות.

עליכם לכתוב תוכנת אסמבלר, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C.

עליכם להגיש את הפריטים הבאים:

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c. או h.).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אובונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים: -Wall -ansi -pedantic. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):

א. קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.

ב. קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

בשל גודל הפרויקט, עליכם לחלק את התוכנית למספר קבצי מקור, לפי משימות. יש להקפיד שקוד המקור של התוכנית יעמוד בקריטריונים של בהירות, קריאות וכתיבה נאה ומובנית.

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

1. הפשטה של מבני הנתונים: רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.

2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לערוך את הקוד באופן מסודר: הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.

3. תיעוד: יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט טובה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את כל הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

על המטלה להיות **מקורית לחלוטין**: אין להיעזר בספריות חיצוניות מלבד הספריות הסטנדרטיות, וכמובן לא בקוד ולא בחלקי קוד הנמצאים ברשת, במקור חיצוני וכו'.

מומלץ לעבוד בזוגות. אין לעבוד בצוותים גדולים יותר. **פרויקט שיוגש על ידי שלשה או יותר, לא ייבדק ולא יקבל ציון**. חובה שסטודנטים, הבוחרים להגיש יחד את הפרויקט, יהיו **שייכים לאותה קבוצת הנחיה**. הציון יהיה זהה לשני הסטודנטים.

מומלץ לקרוא את הגדרת הפרויקט פעם ראשונה ברצף, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא שוב בצורה מעמיקה יותר.

רקע כללי ומטרת הפרויקט

כידוע, קיימות שפות תכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לרוץ באותו מחשב עצמו. כיצד "מכיר" המחשב כל כך הרבה שפות? התשובה פשוטה: המחשב מכיר למעשה שפה אחת בלבד: הוראות ונתונים הכתובים בקוד בינארי. קוד זה מאוחסן בגוש בזיכרון, ונראה כמו רצף של ספרות בינאריות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הרצף הזה לקטעים קטנים בעלי משמעות: הוראות, מענים ונתונים.

למעשה, זיכרון המחשב כולו הוא אוסף של סיביות, שנוהגים לראותן כמקובצות ליחידות בעלות אורך קבוע (בתים, מילים). לא ניתן להבחין, בעין שאינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת ברגיסטרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב.
דוגמאות: העברת מספר מתא בזיכרון לרגיסטר ביע"מ או בחזרה, הוספת 1 למספר הנמצא ברגיסטר, בדיקה האם מספר המאוחסן ברגיסטר שווה לאפס, חיבור וחיסור בין שני רגיסטרים, וכד'.
הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזיכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה.** זהו רצף של ביטים, המהווים קידוד בינארי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד כזה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי** (assembly language) היא שפת תכנות מאפשרת לייצג את הוראות המכונה בצורה סימבולית קלה ונוחה יותר לשימוש. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כלי שנקרא **אסמבלר** (assembler).

כידוע, לכל שפת תכנות עילית יש מהדר (compiler) , או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסמבלר משמש בתפקיד דומה עבור שפת אסמבלי.

לכל מודל של יע"מ (כלומר לכל אירגון של מחשב) יש שפת מכונה ייעודית משלו, ובהתאם גם שפת אסמבלי ייעודית משלו. לפיכך, גם האסמבלר (כלי התרגום) הוא ייעודי ושונה לכל יע"מ.

תפקידו של האסמבלר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסוק בממ"ן זה.

המשימה בפרויקט זה היא לכתוב אסמבלר (כלומר תוכנית המתרגמת לשפת מכונה), עבור שפת אסמבלי שנגדיר כאן במיוחד לצורך הפרויקט.

לתשומת לב: בהסברים הכלליים על אופן עבודת תוכנת האסמבלר, תהיה מדי פעם התייחסות גם לעבודת שלבי הקישור והטעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך

תהליך העיבוד של הפלט של תוכנת האסמבלר. אין לטעות: עליכם לכתוב את תוכנית האסמבלר בלבד. **אין** לכתוב את תוכניות הקישור והטעינה!!!

המחשב הדמיוני ושפת האסמבלי

נגדיר עתה את שפת האסמבלי ואת מודל המחשב הדמיוני, עבור פרויקט זה.

הערה: תיאור מודל המחשב להלן הוא חלקי בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזית), רגיסטרים (אוגרים) וזיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack).

למעבד 8 רגיסטרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7.
גודלו של כל רגיסטר הוא 24 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 23. שמות הרגיסטרים נכתבים תמיד עם אות 'r' קטנה.

כמו כן יש במעבד רגיסטר בשם PSW (program status word), המכיל מספר דגלים המאפיינים את מצב הפעילות במעבד בכל רגע נתון. ראו בהמשך, בתיאור הוראות המכונה, הסברים לגבי השימוש בדגלים אלו.

גודל הזיכרון הוא 2^{21} תאים, בכתובות $0 - 2^{21}-1$, וכל תא הוא בגודל של 24 סיביות. לתא בזיכרון נקרא גם בשם "**מילה**". הסיביות בכל מילה ממוספרות כמו ברגיסטר.

מחשב זה עובד רק עם מספרים שלמים חיוביים ושליליים. אין תמיכה במספרים ממשיים.
האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement).
כמו כן יש תמיכה בתווים (characters), המיוצגים בקוד ascii.

מבנה הוראת מכונה:

כל הוראת מכונה במודל שלנו מורכבת מפעולה ואופרנדים. מספר האופרנדים הוא בין 0 ל-2, בהתאם לסוג הפעולה. מבחינת התפקיד של כל אופרנד, נבחין בין אופרנד מקור (source) ואופרנד יעד (destination).

כל הוראת מכונה מקודדת למספר מילות זיכרון רצופות, **החל ממילה אחת ועד למקסימום שלוש מילים,** בהתאם לשיטת המיעון בה נתון כל אופרנד (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבונה האסמבלר, כל מילה תקודד בבסיס הקסאדצימלי (ראו פרטים לגבי קבצי פלט בהמשך).

בכל סוגי הוראות המכונה, **המבנה של המילה הראשונה תמיד זהה.**
מבנה המילה הראשונה בהוראה הוא כדלהלן:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						מיעון מקור		רגיסטר מקור			מיעון יעד		רגיסטר יעד			funct					A	R	E

במודל המכונה שלנו יש 16 פעולות, בפועל, למרות שניתן לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסמבלי באופן סימבולי על ידי **שם-פעולה**, ובקוד המכונה על ידי קומבינציה ייחודית של ערכי שני שדות במילה הראשונה של ההוראה: **קוד-הפעולה (opcode)**, ו**פונקציה (funct)**.

להלן טבלת הפעולות:

שם הפעולה	funct	קוד-הפעולה (בבסיס עשרוני)
mov		0
cmp		1
add	1	2
sub	2	2
lea		4
clr	1	5
not	2	5
inc	3	5
dec	4	5
jmp	1	9
bne	2	9
jsr	3	9
red		12
prn		13
rts		14
stop		15

<u>הערה</u>: שם-הפעולה נכתב תמיד באותיות קטנות. פרטים על מהות הפעולות השונות יובאו בהמשך.

להלן מפרט השדות במילה הראשונה בקוד המכונה של כל הוראה.

סיביות 18-23: סיביות אלה מכילות את **קוד-הפעולה** (opcode). ישנן מספר פעולות עם קוד פעולה זהה (ראו בטבלה לעיל, קודי-פעולה 2, 5 או 9), ומה שמבדיל ביניהן הוא השדה funct.

סיביות 3-7: שדה זה, הנקרא **funct**, מתפקד כאשר מדובר בפעולה שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קודי-פעולה 2, 5 או 9). השדה funct יכיל ערך ייחודי לכל פעולה מקבוצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש לפעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

סיביות 16-17: מכילות את מספרה של שיטת המיעון של אופרנד המקור.
אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות.
מפרט של שיטות המיעון השונות יינתן בהמשך.

סיביות 13-15: מכילות את מספרו של רגיסטר המקור, במקרה שאופרנד המקור הוא רגיסטר.
אחרת, סיביות אלה יהיו מאופסות

סיביות 11-12: מכילות את מספרה של שיטת המיעון של אופרנד היעד.
אם אין בהוראה אופרנד יעד, סיביות אלה יהיו מאופסות.

סיביות 8-10: מכילות את מספרו של רגיסטר היעד, במקרה שאופרנד היעד הוא רגיסטר.
אחרת סיביות אלה יהיו מאופסות

סיביות 0-2 (השדה 'A,R,E'): אפיון משמעותו של שדה זה בקוד המכונה יובא בהמשך.
במילה <u>הראשונה</u> של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות (R,E) מאופסות.

<u>לתשומת לב</u>: **השדה 'A,R,E' מתווסף לכל אחת מהמילים בקידוד ההוראה** (ראו המפרט של שיטות המיעון בהמשך).

שיטות מיעון:

בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3.
השימוש בחלק משיטות המיעון מצריך מילות-מידע נוספות בקוד המכונה של הוראת המכונה, בנוסף למילה הראשונה.

לכל אופרנד של ההוראה נדרשת **לכל היותר מילת-מידע אחת נוספת**. כאשר בהוראה יש שני אופרנדים הדורשים מילת-מידע נוספת, קודם תופיע מילת-המידע של אופרנד המקור, ולאחריה מילת-המידע של אופרנד היעד.

כל מילת-מידע נוספת של ההוראה מקודדת באחד משלשה סוגים של קידוד. **סיביות 2-0** של כל מילת-מידע הן השדה 'A,R,E', המציין מהו סוג הקידוד של המילה. לכל סוג קידוד יש סיבית נפרדת, שערכה 1 אם מילת-המידע נתונה בסוג קידוד זה, ואחרת ערך הסיבית הוא 0.

- סיבית 2 (הסיבית A) מציינת שקידוד המילה הוא מוחלט (Absolute), ואינו מצריך שינוי בשלבי הקישור והטעינה.
- סיבית 1 (הסיבית R) מציינת שהקידוד הוא של כתובת פנימית הניתנת להזזה (Relocatable), ומצריך שינוי בשלבי הקישור והטעינה.
- סיבית 0 (הסיבית E) מציינת שהקידוד הוא של כתובת חיצונית (External), ומצריך שינוי בשלבי הקישור והטעינה.

הסבר על התפקיד של השדה 'A,R,E' בקוד המכונה יבוא בהמשך.
ערך השדה 'A,R,E' הנדרש בכל אחת משיטות המיעון מופיע בתיאור שיטות המיעון להלן.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 21 סיביות, השוכן בסיביות 3-23 של המילה. הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1, r2 בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל רגיסטר r2.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר <u>ללא סימן</u> ברוחב של 21 סיביות, בסיביות 3-23 של מילת המידע. הסיביות 0-2 במילת המידע הן השדה A,R,E. במיעון ישיר, ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 3-23. אם זוהי כתובת שמייצגת שורה בקובץ המקור הנוכחי (כתובת פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זוהי כתובת שמייצגת שורה בקובץ מקור אחר של התוכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data.'או 'string.', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנחית 'extern.'. התווית מייצגת באופן סימבולי כתובת בזיכרון.	השורה הבאה מגדירה את התווית x: x: .data 23 ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x). <u>דוגמה נוספת</u>: ההוראה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). הכתובת next תקודד בסיביות 3-23 של מילת המידע הנוספת.
2	מיעון יחסי	שיטה זו רלוונטית <u>אך ורק</u> להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בקודי-הפעולה הבאים בלבד: jmp, bne, jsr. <u>לא ניתן</u> להשתמש בשיטה זו בהוראות עם קודי-פעולה אחרים. בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת המכילה את מרחק הקפיצה, במילות זיכרון, מכתובת ההוראה הנוכחית (פקודת הקפיצה) אל כתובת ההוראה המבוקשת (ההוראה הבאה לביצוע). מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 21 סיביות, השוכן בסיביות 3-23 של מילת המידע הנוספת. מרחק זה יהיה שלילי במקרה שהקפיצה היא אל הוראה שבכתובת יותר נמוכה, וחיובי במקרה שהקפיצה היא אל הוראה שבכתובת יותר גבוהה. הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון יחסי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו & ולאחריו ובצמוד אליו מופיע שם של תווית. התווית מייצגת באופן סימבולי כתובת של <u>הוראה בקובץ המקור הנוכחי של התוכנית</u>. ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה. יודגש כי בשיטת מיעון יחסי <u>לא ניתן</u> להשתמש בתווית (כתובת) שמוגדרת בקובץ מקור אחר (כתובת חיצונית).	jmp &next בדוגמה זו, ההוראה מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). נניח כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני). מרחק הקפיצה אל ההוראה בכתובת next הוא 200-, ומרחק זה יקודד בסיביות 3-23 של מילת המידע הנוספת.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
3	מיעון רגיסטר ישיר	האופרנד הוא רגיסטר. לשיטת מיעון זו אין מילת מידע נוספת. מספרו של הרגיסטר מקודד במילה הראשונה של ההוראה, בשדה המתאים: רגיסטר מקור/יעד.	האופרנד הוא שם של רגיסטר.	clr r1 בדוגמה זו, ההוראה clr מאפסת את תוכן הרגיסטר r1.

מפרט הוראות המכונה:

בתיאור הוראות המכונה נשתמש במונח **PC** (קיצור של "**Program Counter**").
זהו רגיסטר פנימי של המעבד (לא רגיסטר כללי), שמכיל בכל רגע נתון את כתובת הזיכרון בה נמצאת **ההוראה הנוכחית שמתבצעת** (הכוונה תמיד לכתובת המילה הראשונה של ההוראה).

הוראות המכונה מתחלקות לשלוש קבוצות, לפי מספר האופרנדים הדרוש לפעולה.

קבוצת ההוראות הראשונה:
אלו הן הוראות הדורשות שני אופרנדים.

ההוראות השייכות לקבוצה זו הן: mov, cmp, add, sub, lea

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0		מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל רגיסטר r1.
cmp	1		מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") ברגיסטר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אזי הדגל Z ("דגל האפס") ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	2	1	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	רגיסטר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.
sub	2	2	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	רגיסטר r1 מקבל את תוצאת החיסור של הקבוע 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	4		lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לרגיסטר r1.

קבוצת ההוראות השניה:
אלו הן הוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. השדות של אופרנד המקור (סיביות 13-17) במילה הראשונה בקידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: clr, not, inc, dec, jmp, bne, jsr, red, prn

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	1	איפוס תוכן האופרנד	clr r2	הרגיסטר r2 מקבל את הערך 0.
not	5	2	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט ברגיסטר r2 מתהפך.
inc	5	3	הגדלת תוכן האופרנד באחד.	inc r2	תוכן הרגיסטר r2 מוגדל ב-1.
dec	5	4	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב- 1.
jmp	9	1	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp &Line	PC←PC+distanceTo(Line) מצביע התכנית מקבל את המען שמחושב על ידי חיבור המרחק לתווית Line עם מען ההוראה הנוכחית, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
bne	9	2	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. אם ערכו של הדגל Z ברגיסטר הסטטוס (PSW) הינו 0, אזי מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הוא 0, אזי PC ← address(Line) מצביע התכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
jsr	9	3	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. <u>הערה</u>: חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC ← address(SUBR) מצביע התכנית יקבל את כתובת התווית SUBR, ולפיכך, ההוראה הבאה שתתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לרגיסטר r1.
prn	13		הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	יודפס לפלט התו (קוד ascii) הנמצא ברגיסטר r1

קבוצת ההוראות השלישית:
אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 8-17) במילה הראשונה של קידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: rts, stop.

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
rts	14	מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC). <u>הערה</u>: ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת jsr	rts	PC ← pop() ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה.
stop	15	עצירת ריצת התוכנית.	stop	התוכנית עוצרת מיידית.

<u>מבנה שפת האסמבלי:</u>

תכנית בשפת אסמבלי בנויה **<u>ממאקרואים</u>** **<u>וממשפטים</u>** (statements).

<u>מאקרואים</u>:

מאקרואים הם קטעי קוד הכוללים בתוכם משפטים. בתוכנית ניתן להגדיר מאקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במאקרו ממקום מסוים בתוכנית יגרום לפרישת המאקרו לאותו מקום.

הגדרת מאקרו נעשית באופן הבא: (בדוגמה שם המאקרו הוא a_mc)

```
mcro  a_mc
  inc r2
  mov  A,r1
mcroend
```

שימוש במאקרו הוא פשוט אזכור שמו.
למשל, אם בתוכנית במקום כלשהו כתוב:

```
.
.
a_mc
.
.
a_mc
.
.
```

בדוגמה זו, השתמשנו פעמיים במאקרו a_mc, התוכנית לאחר פרישת המאקרו תיראה כך:

```
.
.
  inc r2
  mov  A,r1
.
.
  inc r2
  mov  A,r1
.
.
```

<u>התוכנית לאחר פרישת המאקרו היא התוכנית שהאסמבלר אמור לתרגם.</u>

<u>הנחות והנחיות לגבי מאקרו:</u>

- אין במערכת הגדרות מאקרו מקוננות (אין צורך לבדוק זאת).
- שם של הוראה או הנחיה לא יכול להיות שם של מאקרו (יש לבדוק זאת).
- ניתן להניח שלכל שורת מאקרו בקוד המקור קיימת סגירה עם שורת mcroend (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקריאה למאקרו (אין צורך לבדוק זאת).
- נדרש שהקדם-אסמבלר ייצור קובץ עם הקוד המורחב הכולל פרישה של המאקרו (הרחבה של קובץ המקור המתואר בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המאקרו, לעומת "קובץ מקור ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרת המאקרואים.

<u>לסיכום, במאקרו יש לבדוק:</u>

(1) שם המאקרו תקין (אינו שם הוראה וכדומה)

(2) בשורת ההגדרה ובשורת הסיום אין תווים נוספים

אם נמצאה שגיאה בשלב פרישת המאקרו - אי אפשר לעבור לשלבים הבאים:

יש לעצור להודיע על השגיאות ולעבור לקובץ המקור הבא (אם קיים).

הערה: שגיאות **בגוף** המאקרו (אם יש) מגלים בשלבים הבאים.

<u>משפטים</u>:

קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע <u>בשורה נפרדת</u>. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו '\n' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תווים לכל היותר (לא כולל התו \n).

יש ארבעה סוגי משפטים (שורות בקובץ המקור) בשפת אסמבלי, והם:

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק את התווים ' ' ו- '\t' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התכנית. המשפט מורכב משם של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

<u>משפט הנחיה:</u>

משפט הנחיה הוא בעל המבנה הבא:

בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי, שיתואר בהמשך. התווית היא אופציונלית.

לאחר מכן מופיע שם ההנחיה. לאחר שם ההנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנחיה). שם של הנחיה מתחיל בתו '.' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

יש לשים לב: למילים בקוד המכונה הנוצרות <u>ממשפט הנחיה</u> לא מצורף השדה A,R,E, והקידוד ממלא את כל הסיביות של המילה.

יש ארבעה סוגים של משפטי הנחיה, והם:

1. ההנחיה 'data.'

הפרמטרים של ההנחיה 'data.' הם מספרים שלמים חוקיים (אחד או יותר) המופרדים על ידי התו ',' (פסיק). לדוגמה:

.data 7, −57, +17, 9

יש לשים לב שהפסיקים אינם חייבים להיות צמודים למספרים. בין מספר לפסיק ובין פסיק למספר יכולים להופיע רווחים וטאבים בכל כמות (או בכלל לא), אולם הפסיק חייב להופיע בין המספרים. כמו כן, אסור שיופיע יותר מפסיק אחד בין שני מספרים, וגם לא פסיק אחרי המספר האחרון או לפני המספר הראשון.

המשפט 'data.' מנחה את האסמבלר להקצות מקום בתמונת הנתונים (data image), אשר בו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בהנחית data. מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונת הנתונים דרך שם התווית (למעשה, זוהי דרך להגדיר שם של משתנה).

כלומר אם נכתוב:

XYZ: .data 7, –57, +17, 9

אזי יוקצו בתמונת הנתונים ארבע מילים רצופות שיכילו את המספרים שמופיעים בהנחיה. התווית XYZ מזוהה עם כתובת המילה הראשונה.

אם נכתוב בתכנית את ההוראה:

mov XYZ, r1

אזי בזמן ריצת התכנית יוכנס לרגיסטר r1 הערך 7.

ואילו ההוראה:

lea XYZ, r1

תכניס לרגיסטר r1 את ערך התווית XYZ (כלומר הכתובת בזיכרון בה מאוחסן הערך 7).

2. ההנחיה 'string.'

להנחיה 'string.' פרמטר אחד, שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים, ומוכנסים אל תמונת הנתונים לפי סדרם, כל תו במילה נפרדת. בסוף המחרוזת יתווסף התן '\0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה למה שנעשה עבור 'data.' (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחרוזת).

לדוגמה, ההנחיה:

```
STR:  .string "abcdef"
```

מקצה בתמונת הנתונים רצף של 7 מילים, ומאתחלת את המילים לקודי ה-ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובת התחלת המחרוזת.

3. ההנחיה 'entry.'

להנחיה 'entry.' פרמטר והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת ההנחיה entry. היא לאפיין את התווית הזו באופן שיאפשר לקוד אסמבלי הנמצא בקבצי מקור אחרים להשתמש בה (כאופרנד של הוראה).

לדוגמה, השורות:

```
        .entry   HELLO
HELLO:  add      #1, r1
        ........
```

מודיעות לאסמבלר שאפשר להתייחס בקובץ אחר לתווית HELLO המוגדרת בקובץ הנוכחי.

<u>לתשומת לב</u>: תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

4. ההנחיה 'extern.'

להנחיה 'extern.' פרמטר והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמבלר כי התווית מוגדרת בקובץ מקור אחר, וכי קוד האסמבלי בקובץ הנוכחי עושה בתווית שימוש.

נשים לב כי הנחיה זו תואמת להנחית 'entry.' המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תתבצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשתמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לממ"ן זה).

לדוגמה, משפט ההנחיה 'extern.' התואם למשפט ההנחיה 'entry' מהדוגמה הקודמת יהיה:

```
.extern   HELLO
```

<u>הערה</u>: לא ניתן להגדיר באותו הקובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO).

<u>לתשומת לב</u>: תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

<u>משפט הוראה</u>:

משפט הוראה מורכב מהחלקים הבאים:
1. תווית אופציונלית.
2. שם הפעולה.
3. אופרנדים, בהתאם לסוג הפעולה (בין 0 ל-2 אופרנדים).

אם מוגדרת תווית בשורת ההוראה, אזי היא תוכנס אל טבלת הסמלים. ערך התווית יהיה מען המילה הראשונה של ההוראה בתוך תמונת הקוד שבונה האסמבלר.

שם הפעולה תמיד באותיות קטנות (lower case), והוא אחת מ- 16 הפעולות שפורטו לעיל.

לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). בדומה להנחיה 'data.', **לא חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאבים משני צידי הפסיק היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

label: opcode source-operand, target-operand

לדוגמה:

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label: opcode target-operand

לדוגמה:

HELLO: bne &XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label: opcode

לדוגמה:

END: stop

<u>אפיון השדות במשפטים של שפת האסמבלי</u>

<u>תווית</u>:
בתיאור שיטות המיעון למעלה הסברנו כי תווית היא ייצוג סימבולי של כתובת בזיכרון. נרחיב כאן את ההסבר:
תווית היא למעשה סמל שמוגדר בתחילת משפט הוראה, או בתחילת הנחיית data. או string.
תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

<u>הגדרה של תווית</u> מסתיימת בתו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ':' חייב להיות צמוד לתווית (ללא רווחים).

אסור שאותה תווית תוגדר יותר מפעם אחת (כמובן בשורות שונות). אותיות קטנות וגדולות נחשבות <u>שונות זו מזו</u>.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של רגיסטר) אינן יכולות לשמש גם כשם של תווית. כמו כן, אסור שאותו סמל ישמש הן כתווית והן כשם של מאקרו (יש לבדוק זאת).

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. תווית המוגדרת בהנחיות data., string., תקבל את ערך מונה הנתונים (data counter) הנוכחי, בעוד שתווית המוגדרת בשורת הוראה תקבל את ערך מונה ההוראות (instruction counter) הנוכחי.

לתשומת לב: מותר במשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנחיית extern. כלשהי בקובץ הנוכחי).

מספר:

מספר חוקי מתחיל בסימן אופציונלי: '–' או '+' ולאחריו סדרה כלשהי של ספרות בבסיס עשרוני. לדוגמה: 76, 5–, 123+ הם מספרים חוקיים. אין תמיכה בשפת האסמבלי שלנו בייצוג בבסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

מחרוזת:

מחרוזת חוקית היא סדרת תווי ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפולות (המרכאות אינן נחשבות חלק מהמחרוזת). דוגמה למחרוזת חוקית: "hello world".

סימון המילים בקוד המכונה באמצעות המאפיין "A,R,E"

בכל מילה בקוד המכונה של הוראה (לא של נתונים), האסמבלר מכניס מידע עבור תהליך הקישור והטעינה. זה השדה A,R,E. המידע ישמש לתיקונים בקוד בכל פעם שייטען לזיכרון לצורך הרצה. האסמבלר בונה מלכתחילה קוד שמיועד לטעינה החל מכתובת ההתחלה. התיקונים יאפשרו לטעון את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

שלוש הסיביות בשדה A,R,E יכילו ערכים בינאריים כפי שהוסבר בתיאור שיטות המיעון. המשמעות של כל ערך מפורטת להלן.

האות 'A' (קיצור של absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופרנד מיידי).

האות 'R' (קיצור של relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה כתובת של תווית המוגדרת בקובץ המקור).

האות 'E' (קיצור של external) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (external) (למשל מילה המכילה כתובת של תווית חיצונית, כלומר תווית שאינה מוגדרת בקובץ המקור).

כאשר האסמבלר מקבל כקלט תוכנית בשפת אסמבלי, עליו לטפל תחילה בפרישת המאקרואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המאקרואים. כלומר, פרישת המאקרואים תעשה בשלב "קדם אסמבלר", לפני שלב האסמבלר (המתואר בהמשך) .
אם התכנית אינה מכילה מאקרו, תוכנית הפרישה תהיה זהה לתכנית המקור.

דוגמה לשלב קדם אסמבלר. האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

```
MAIN:     add     r3, LIST
LOOP:     prn     #48
          mcro  a_mc
          cmp     K, #-6
          bne     &END
          mcroend
          lea     STR, r6
          inc     r6
          mov     r3, K
          sub     r1, r4
          bne     END
          a_mc
          dec     K
          jmp     &LOOP
END:      stop
STR:     .string  "abcd"
LIST:    .data    6, -9
         .data   -100
K:       .data    31
```

תחילה האסמבלר עובר על התוכנית ופורש את כל המאקרואים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבור לשלב הבא. אחרת, יש להציג את השגיאות ולא לייצר קבצים. בדוגמה זו, התוכנית לאחר פרישת המאקרו תיראה כך:

```
MAIN:     add     r3, LIST
LOOP:     prn     #48
          lea     STR, r6
          inc     r6
          mov     r3, K
          sub     r1, r4
          bne     END
          cmp     K, #-6
          bne     &END
          dec     K
          jmp     &LOOP
END:      stop
STR:     .string  "abcd"
LIST:    .data    6, -9
         .data   -100
K:       .data    31
```

קוד התכנית, לאחר הפרישה, ישמר בקובץ חדש, כפי שיוסבר בהמשך.

אלגוריתם שלדי של קדם האסמבלר

נציג להלן אלגוריתם שלדי לתהליך קדם האסמבלר. לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עבור ל- 9 (סיום).
2. האם השדה הראשון הוא שם מאקרו המופיע בטבלת המאקרו (כגון a_mc)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל 1. אחרת, המשך.
3. האם השדה הראשון הוא " **mcro** " (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "יש **mcro** ".
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה a_mc).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).
   אם דגל "יש **mcro** " דולק ולא זוהתה תווית **mcroend** הכנס את השורה לטבלת המאקרו ומחק את השורה הנ"ל מהקובץ. אחרת (לא מאקרו) חזור ל 1.
7. האם זוהתה תווית **mcroend**? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל "יש **mcro** ". חזור ל- 1. (סיום שמירת הגדרת מאקרו).
9. סיום: שמירת קובץ מאקרו פרוש.

אסמבלר עם שני מעברים

במעבר הראשון של האסמבלר, יש לזהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מספרי שהוא המען בזיכרון שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספרי הרגיסטרים, בונים את קוד המכונה.

עליו להחליף את שמות הפעולות בקוד הבינארי השקול להם במודל המחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את כל הסמלים (למשל LIST, MAIN) במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטען בזיכרון החל ממען 100 (בבסיס 10) . במקרה זה נקבל את ה"תרגום" הבא:

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000001111111010
0000102 0000103	LOOP: prn #48	 Immediate value 48	001101000000000000000100 000000000000000110000100
0000104 0000105	lea STR, r6	 Address of label STR	000100010001111000000100 000000000000001111010010
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	 Address of label K	000000110110100000000100 000000000000010000010010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	 Address of label END	001001000000100000010100 000000000000001111001010
0000112 0000113 0000114	cmp K, #-6	 Address of label K Immediate value -6	000001010000000000000100 000000000000010000010010 111111111111111111010100
0000115 0000116	bne &END	 Distance to label END	001001000001000000010100 000000000000000000110100
0000117 0000118	dec K	 Address of label K	000101000000100000100100 000000000000010000010010
0000119 0000120	jmp &LOOP	 Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121	END: stop		001111000000000000000100
0000122	STR: .string “abcd”	Ascii code ‘a’	000000000000000001100001
0000123		Ascii code ‘b’	000000000000000001100010
0000124		Ascii code ‘c’	000000000000000001100011
0000125		Ascii code ‘d’	000000000000000001100100
0000126		Ascii code ‘\0’	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000011111

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של ההוראות והקודים הבינאריים המתאימים להם, ולכן שמות הפעולות ניתנים להמרה לבינארי בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצוא את הקוד הבינארי.

כדי לבצע המרה לבינארי של אופרנדים שכתובים בשיטות מיעון המשתמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END משויך למען 121 (עשרוני), ושהסמל K משויך למען 130, אלא רק לאחר שנקראו כל שורות התכנית.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים, עם הערכים המספריים המשויכים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של הוראות התוכנית, בערכיהם המספריים. הביצוע של שני שלבים אלה כרוך בשתי סריקות (הנקראות ״מעברים״) של קובץ המקור.

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משויך ערך מספרי, שהוא מען בזיכרון. בדוגמה לעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	102
END	121
STR	122
LIST	127
K	130

במעבר השני נעשית ההמרה של קוד המקור לקוד מכונה. בתחילת המעבר השני צריכים הערכים של הסמלים להיות כבר ידועים.

לתשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התכנית טרם מוכנה לטעינה לזיכרון לצורך ביצוע. קוד המכונה חייב לעבור לשלבי הקישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה **אינם** חלק מהממ"ן).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויך לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממען 100. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבלר טבלה, שיש בה קוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסמבלר כל שם פעולה בקוד שלה, וכן כל אופרנד מוחלף בקידוד מתאים, אך פעולת החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מיעון מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעויות שונות, בכל אחת משיטות המיעון, ולכן יתאימו לה קידודים שונים לפי שיטות המיעון. לדוגמה, פעולת ההזזה mov יכולה להתייחס להעתקת תוכן תא זיכרון לרגיסטר, או להעתקת תוכן רגיסטר לרגיסטר אחר, וכן הלאה. לכל אפשרות כזאת של mov עשוי להתאים קידוד שונה.

על האסמבלר לסרוק את שורת ההוראה בשלמותה, ולהחליט לגבי הקידוד לפי האופרנדים. בדרך כלל מתחלק הקידוד לשדה של שם הפעולה, ושדות נוספים המכילים מידע לגבי שיטות המיעון. כל השדות ביחד דורשים מילה אחת או יותר בקוד המכונה.

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הנוכחי של ה-IC. כך מקבלות כל התוויות את מעניהן בעת ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסמבלר לשלוף את המען המתאים מטבלת הסמלים.

הוראה יכולה להתייחס גם לסמל שטרם הוגדר עד כה בתכנית, אלא יוגדר רק בהמשך התכנית. להלן לדוגמה, הוראת הסתעפות למען שמוגדר על ידי התווית A שמופיעה רק בהמשך הקוד:

```
        bne   A
        .
        .
        .
A:      ......
```

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמובן לא יודע את המען המשויך לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מיידי, או רגיסטר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות data., string.).

המעבר השני

ראינו שבמעבר הראשון, האסמבלר אינו יכול לבנות את קוד המכונה של אופרנדים המשתמשים בסמלים שעדיין לא הוגדרו. רק לאחר שהאסמבלר עבר על כל התכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יכול האסמבלר להשלים את קוד המכונה של כל האופרנדים.

לשם כך מבצע האסמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכן את קוד המכונה של האופרנדים המשתמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים.
בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

הפרדת הוראות ונתונים

בתכנית מבחינים בשני סוגים של תוכן: הוראות ונתונים. יש לארגן את קוד המכונה כך שתהיה הפרדה בין הנתונים וההוראות. הפרדת ההוראות והנתונים לקטעים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשתמשות בהן.

אחת הסכנות הטמונות באי הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו הסתעפות לא נכונה. התכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסמבלר שלנו חייב להפריד, בקוד המכונה שהוא מיצר, בין קטע הנתונים לקטע ההוראות. **כלומר בקובץ הפלט (בקוד המכונה) תהיה הפרדה של הוראות ונתונים לשני קטעים נפרדים, ואילו בקובץ הקלט אין חובה שתהיה הפרדה כזו.** בהמשך מתואר אלגוריתם של האסמבלר, ובו פרטים כיצד לבצע את ההפרדה.

גילוי שגיאות בתכנית המקור

הנחת המטלה היא **שאין שגיאות בהגדרות המאקרו**, ולכן שלב קדם האסמבלר אינו מכיל שלב גילוי שגיאות. אין גם צורך לבדוק שגיאות בפתיחת / סגירת המאקרו (למשל אם המאקרו לא מסתיים – ניתן להניח שהנ"ל תקין). לעומת זאת, האסמבלר אמור לגלות ולדווח על **שגיאות בתחביר של תוכנית המקור**, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים לפעולה, שם רגיסטר לא קיים, ועוד שגיאות אחרות. כמו כן מוודא האסמבלר שכל סמל מוגדר פעם אחת בדיוק.

מכאן, שכל שגיאה המתגלה על ידי האסמבלר נגרמת (בדרך כלל) על ידי שורת קלט מסוימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד יחיד, האסמבלר ייתן הודעת שגיאה בנוסח "יותר מדי אופרנדים".

הערה: אם יש שגיאה בקוד האסמבלי בגוף מאקרו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המאקרו. נשים לב שכאשר האסמבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש ממאקרו, כך שלא ניתן לחסוך גילויי שגיאה כפולים.

האסמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי stdout. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה (מניין השורות בקובץ מתחיל ב-1).

לתשומת לב: האסמבלר אינו עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבור על הקלט כדי לגלות שגיאות נוספות, ככל שישנן. כמובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ממילא אי אפשר להשלים את קוד המכונה).

הטבלה הבאה מפרטת מהן של שיטות המיעון החוקיות, עבור אופרנד המקור ואופרנד היעד של ההוראות השונות הקיימות בשפה הנתונה:

Opcode	funct	שם ההוראה	שיטות מיעון חוקיות עבור אופרנד המקור	שיטות מיעון חוקיות עבור אופרנד היעד
0		mov	0,1,3	1,3
1		cmp	0,1,3	0,1,3
2	1	add	0,1,3	1,3
2	2	sub	0,1,3	1,3
4		lea	1	1,3
5	1	clr	אין אופרנד מקור	1,3
5	2	not	אין אופרנד מקור	1,3
5	3	inc	אין אופרנד מקור	1,3
5	4	dec	אין אופרנד מקור	1,3
9	1	jmp	אין אופרנד מקור	1,2
9	2	bne	אין אופרנד מקור	1,2
9	3	jsr	אין אופרנד מקור	1,2
12		red	אין אופרנד מקור	1,3
13		prn	אין אופרנד מקור	0,1,3
14		rts	אין אופרנד מקור	אין אופרנד יעד
15		stop	אין אופרנד מקור	אין אופרנד יעד

אלגוריתם שלדי של האסמבלר

לחידוד ההבנה של תהליך העבודה של האסמבלר, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני. לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

אנו מחלקים את קוד המכונה לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data). לכל אזור יש מונה משלו, ונסמנם IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter). נבנה את קוד המכונה כך שיתאים לטעינה לזיכרון **החל מכתובת 100**.

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 100.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-17.
3. האם השדה הראשון בשורה הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data. או string.? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין data. ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, ועדכן את מונה הנתונים DC בהתאם לאורכם. חזור ל-2.
8. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-11.

9. אם זוהי הנחית entry. חזור ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הנחית extern., הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין external. חזור ל-2.
11. זוהי שורת הוראה. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו של הסמל יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכולל שתופסת ההוראה בקוד המכונה (נקרא למספר זה L).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת-מידע נוספת המקודדת אופרנד במיעון מיידי.
15. שמור את הערכים IC ו- L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן IC ← IC + L, וחזור ל-2.
17. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
18. שמור את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו- DCF). נשתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין כ- data , ע"י הוספת הערך ICF (ראה הסבר לכך בהמשך).
20. התחל מעבר שני.

מעבר שני

1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 7.
2. אם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
3. האם זוהי הנחית data. או string. או extern. ? אם כן, חזור ל- 1.
4. האם זוהי הנחית entry. ? אם לא, עבור ל- 6.
5. הוסף בטבלת הסמלים את המאפיין entry למאפייני הסמל המופיע כאופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל- 1.
6. השלם את הקידוד הבינארי של מילות-המידע של האופרנדים, בהתאם לשיטות המיעון שבשימוש. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה). אם הסמל מאופיין external, הוסף את כתובת מילת-המידע הרלוונטית לרשימת מילות-מידע שמתייחסות לסמל חיצוני. לפי הצורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים IC ו-L של ההוראה, כפי שנשמרו במעבר הראשון. חזור ל- 1.
7. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו למעלה (**לאחר שלב פרישת המאקרואים**), ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה:

```
MAIN:    add     r3, LIST
LOOP:    prn     #48
         lea     STR, r6
         inc     r6
         mov     r3,K
         sub     r1, r4
         bne     END
         cmp     K, #-6
         bne     &END
         dec     K
         jmp     &LOOP
END:     stop
STR:     .string "abcd"
LIST:    .data   6, -9
         .data   -100
K:       .data   31
```

נבצע עתה מעבר ראשון על הקוד הנתון. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. כמו כן, נקודד מילות-מידע נוספות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את החלקים שעדיין לא מתורגמים במעבר זה, נשאיר כמות שהם (מסומנים ב- ? בדוגמה להלן).

Address (decimal)	**Source Code**	**Explanation**	**Binary Machine Code**
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 ?
0000102 0000103	LOOP: prn #48	 Immediate value 48	001101000000000000000100 000000000000000110000100
0000104 0000105	lea STR, r6	 Address of label STR	000100010001111000000100 ?
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	 Address of label K	000000110110100000000100 ?
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	 Address of label END	001001000000100000010100 ?
0000112 0000113 0000114	cmp K, #-6	 Address of label K Immediate value -6	000000101000000000000100 ? 111111111111111111010100
0000115 0000116	bne &END	 Distance to label END	001001000001000000010100 ?
0000117 0000118	dec K	 Address of label K	000101000000100000100100 ?
0000119 0000120	jmp &LOOP	 Distance to label LOOP	001001000001000000001100 **?**
0000121	END: stop		001111000000000000000100
0000122	STR: .string “abcd”	Ascii code ‘a’	000000000000000001100001
0000123		Ascii code ‘b’	000000000000000001100010
0000124		Ascii code ‘c’	000000000000000001100011
0000125		Ascii code ‘d’	000000000000000001100100
0000126		Ascii code ‘\0’	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000011111

טבלת הסמלים אחרי המעבר ראשון היא:

סמל	ערך (בבסיס עשרוני)	איפיון הסמל
MAIN	100	code
LOOP	102	code
END	121	code
STR	122	data
LIST	127	data
K	130	data

נבצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במילים המסומנות "?".
הקוד הבינארי בצורתו הסופית כאן זהה לקוד שהוצג בתחילת הנושא "**אסמבלר עם שני מעברים**".

הערה: כאמור, האסמבלר בונה קוד מכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100 (עשרוני).
אם הטעינה בפועל (לצורך הרצת התוכנית) תהיה לכתובת אחרת, יידרשו תיקונים בקוד הבינארי בשלב הטעינה, שיוכנסו בעזרת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	**Source Code**	**Explanation**	**Binary Machine Code**
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000001111111010
0000102 0000103	LOOP: prn #48	 Immediate value 48	001101000000000000000100 000000000000000110000100
0000104 0000105	lea STR, r6	 Address of label STR	000100010001111000000100 000000000000001111010010
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	 Address of label K	000000110110100000000100 000000000000010000010010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	 Address of label END	001001000000100000010100 000000000000001111001010
0000112 0000113 0000114	cmp K, #-6	 Address of label K Immediate value -6	000000101000000000000100 000000000000010000010010 111111111111111111010100
0000115 0000116	bne &END	 Distance to label END	001001000001000000010100 000000000000000000110100
0000117 0000118	dec K	 Address of label K	000101000000100000100100 000000000000010000010010
0000119 0000120	jmp &LOOP	 Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121	END: stop		001111000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	000000000000000001100001
0000123		Ascii code 'b'	000000000000000001100010
0000124		Ascii code 'c'	000000000000000001100011
0000125		Ascii code 'd'	000000000000000001100100
0000126		Ascii code '\0'	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000011111

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטעינה.

כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

קבצי קלט ופלט של האסמבלר

בהפעלה של האסמבלר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר).
אלו הם קבצי טקסט, ובהם תוכניות בתחביר של שפת האסמבלי שהוגדרה בממ״ן זה.

האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קבצי פלט כדלקמן:

- קובץ am, המכיל את קובץ המקור לאחר שלב קדם האסמבלר (לאחר פרישת המאקרואים)
- קובץ object, המכיל את קוד המכונה.
- קובץ externals, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל שהוצהר כחיצוני (סמל שהופיע כאופרנד של הנחיית extern., ומאופיין בטבלת הסמלים כ- external).
- קובץ entries, ובו פרטים על כל סמל שמוצהר כנקודת כניסה (סמל שהופיע כאופרנד של הנחיית entry., ומאופיין בטבלת הסמלים כ- entry).

אם אין בקובץ המקור אף הנחיית extern., האסמבלר <u>לא יוצר</u> את קובץ הפלט מסוג externals.
אם אין בקובץ המקור אף הנחיית entry., האסמבלר <u>לא יוצר</u> את קובץ הפלט מסוג entries.

שמות קבצי המקור חייבים להיות עם הסיומת ״as.״. למשל, השמות x.as , y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית <u>ללא ציון הסיומת</u>.

לדוגמה: נניח שתוכנית האסמבלר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

assembler x y hello

תריץ את האסמבלר על הקבצים: x.as, y.as, hello.as.

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת מתאימה: הסיומת ״am.״ עבור קובץ לאחר פרישת מאקרו, הסיומת ״ob.״ עבור קובץ ה-object, הסיומת ״ent.״ עבור קובץ ה-entries, והסיומת ״ext.״ עבור קובץ ה-externals.

לדוגמה, בהפעלת האסמבלר באמצעות שורת הפקודה: assembler x
ייווצר קובץ פלט x.ob, וכן קבצי פלט x.ext ו- x.ent ככל שיש הנחיות entry. או extern. בקובץ המקור.
אם אין מאקרו בקובץ המקור, אזי קובץ ״am.״ יהיה זהה לקובץ ״as.״ .

אופן פעולת האסמבלר

נרחיב כאן על אופן פעולת האסמבלר, בנוסף לאלגוריתם השלדי שניתן לעיל.

האסמבלר מחזיק שני מערכים, שייקראו להלן מערך ההוראות ומערך הנתונים. מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (גודל כל כניסה במערך זהה לגודלה של מילת מכונה (בסיביות). במערך ההוראות מכניס האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות מסוג data. string.).

לאסמבלר יש שני מונים: מונה ההוראות (IC) ומונה הנתונים (DC). מונים אלו מצביעים על המקום הבא הפנוי במערכים לעיל, בהתאמה. כשמתחיל האסמבלר לעבור על קובץ מקור, שני מונים אלו מקבלים ערך התחלתי.

בנוסף יש לאסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על הקובץ. לטבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל (תווית) נשמרים שמו, ערכו, ומאפיינים שונים שצוינו קודם, כגון המיקום (data או code), או אופן העדכון (למשל external).

האסמבלר קורא את קובץ המקור שורה אחר שורה, מחליט מהו סוג השורה (הערה, הוראה, הנחיה, או שורה ריקה) ופועל בהתאם.

1. שורה ריקה או שורת הערה: האסמבלר מתעלם מהשורה ועובר לשורה הבאה.

2. שורת הוראה:

האסמבלר מוצא מהי הפעולה, ומהן שיטות המיעון של האופרנדים. (מספר האופרנדים אותם הוא מחפש נקבע בהתאם להוראה אותה הוא מצא).

אם האסמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזי התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC, והמאפיין הוא code.

האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

- אם זה רגיסטר – האופרנד הוא מספר הרגיסטר.
- אם זו תווית (מיעון ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים).
- אם זה התו # ואחריו מספר– האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תיאור שיטות המיעון לעיל)

קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד, כפי שהוסבר לעיל בהגדרת שיטות המיעון. למשל, מספר מציין מיעון מיידי, תווית מציינת מיעון ישיר וכד'.

לאחר שהאסמבלר ניתח את השורה והחליט לגבי הפעולה, שיטת מיעון אופרנד המקור (אם יש), ושיטת מיעון אופרנד היעד (אם יש), הוא פועל באופן הבא:

אם זוהי פעולה בעלת שני אופרנדים, אזי האסמבלר מכניס למערך ההוראות, במקום עליו מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בשיטת הייצוג של הוראות המכונה כפי שתואר קודם לכן). מילה זו מכילה את קוד הפעולה, ואת שיטות המיעון. בנוסף "משרייין" האסמבלר מקום במערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופרנדים הם בשיטת מיעון רגיסטר או מיידי, האסמבלר מקודד כעת את המילים הנוספות הרלוונטיות במערך ההוראות.

אם זוהי פעולה בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי הקידוד הינו זהה לעיל, פרט לסיביות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכילו תמיד 0, מכיוון שאינן רלוונטיות לפעולה.

אם זוהי פעולה ללא אופרנדים אזי תקודד רק המילה הראשונה (והיחידה). הסיביות של שיטות המיעון של האופרנדים יכילו 0.

אם בשורת ההוראה קיימת תווית, אזי התווית מוכנסת אל טבלת הסמלים תחת השם המתאים, ערך התווית הוא ערך מונה ההוראות לפני קידוד ההוראה.

3. שורת הנחיה:

כאשר האסמבלר קורא בקובץ המקור שורת הנחיה, הוא פועל בהתאם לסוג ההנחיה, באופן הבא:

I. ‘.data’

האסמבלר קורא את רשימת המספרים, המופיעה לאחר ‘.data’, מכניס כל מספר אל מערך הנתונים, ומקדם את מצביע הנתונים DC באחד עבור כל מספר שהוכנס.

אם בשורה ‘.data’ יש תווית, אזי תווית זו מוכנסת לטבלת הסמלים. היא מקבלת את הערך של מונה הנתונים DC שלפני הכנסת המספרים למערך הנתונים. כן מסומן שההגדרה ניתנה בחלק הנתונים.

II. ‘.string’

הטיפול ב-‘.string’ דומה ל- ‘.data’, אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו בכניסה נפרדת). לאחר מכן מוכנס הערך 0 (המציין סוף מחרוזת) אל מערך הנתונים. מונה הנתונים מקודם באורך המחרוזת + 1 (גם האפס בסוף המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בשורה זו זהה לטיפול הנעשה בהנחיה ‘.data’.

III. ‘.entry’

זוהי בקשה לאסמבלר להכניס את התווית המופיעה כאופרנד של ‘.entry’ אל קובץ ה-entries. האסמבלר רושם את הבקשה ובסיום העבודה, התווית הנ״ל תירשם בקובץ ה-entries.

IV. ‘.extern’

זוהי הצהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסמבלי בקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל אל טבלת הסמלים. ערכו הוא 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), וטיפוסו הוא external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל (וגם אין זה משנה עבור האסמבלר).

יש לשים לב: בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחית extern).

בסוף המעבר הראשון, האסמבלר מעדכן בטבלת הסמלים כל סמל המאופיין כ- data, על ידי הוספת IC+100 (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, הנתונים מופרדים מההוראות, וכל הנתונים נדרשים להופיע אחרי כל ההוראות. סמל מסוג data הוא למעשה תווית באזור הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של קידוד כל ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

טבלת הסמלים מכילה כעת את כל הערכים הנחוצים להשלמת הקידוד (למעט ערכים של סמלים חיצוניים).

במעבר השני, האסמבלר מקודד באמצעות טבלת הסמלים את כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. אלו הן מילים שצריכות להכיל כתובות של תוויות.

פורמט קובץ ה- object

קובץ זה מכיל את תמונת הזיכרון של קוד המכונה, בשני חלקים: תמונת ההוראות ראשונה, ואחריה ובצמוד תמונת הנתונים.

כזכור, האסמבלר מקודד את ההוראות כך שתמונת ההוראות תתאים לטעינה החל מכתובת 100 (עשרוני) בזיכרון. נשים לב שרק בסוף המעבר הראשון יודעים מהו הגודל הכולל של תמונת ההוראות. מכיוון שתמונת הנתונים נמצאת אחרי תמונת ההוראות, גודל תמונת ההוראות משפיע על הכתובות בתמונת הנתונים. זו הסיבה שבגללה היה צורך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, בצעד 19 הוספנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד של מילות-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונת הזיכרון.

כעת האסמבלר יכול לכתוב את תמונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה- object).

השורה הראשונה בקובץ ה- object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכולל של תמונת ההוראות (במילות זיכרון), והשני הוא האורך הכולל של תמונת הנתונים (במילות זיכרון). בין שני המספרים מפריד רווח אחד.
כזכור, במעבר הראשון, בצעד 18, נשמרו הערכים ICF ו- IDF. האורך הכולל של תמונת ההוראות הוא 100−ICF, והאורך הכולל של תמונת הנתונים הוא IDF.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה שני שדות: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס <u>עשרוני</u> בשבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם בבסיס <u>הקסאדצימלי</u> ב-6 ספרות (כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד.

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה שם של סמל שהוגדר כ- entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בבסיס עשרוני.

פורמט קובץ ה- externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובת בקוד המכונה בה יש קידוד של אופרנד המתייחס לסמל זה. כמובן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס עשרוני.

<u>לתשומת לב</u>: ייתכן ויש מספר כתובות בקוד המכונה בהן מילות-המידע מתייחסות לאותו סמל חיצוני. לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

<u>נדגים את קבצי הפלט שמייצר האסמבלר עבור קובץ מקור בשם ps.as.</u>
התוכנית לאחר שלב פרישת המאקרו תיראה כך:

```
; file ps.as

.entry LIST
.extern W
MAIN:       add     r3, LIST
LOOP:       prn     #48
            lea     W, r6
            inc     r6
            mov     r3, K
            sub     r1, r4
            bne     END
            cmp     K, #-6
            bne     &END
            dec     W
.entry MAIN
            jmp     &LOOP
            add     L3, L3
END:        stop

STR:        .string "abcd"
LIST:       .data   6, -9
            .data   -100
K:          .data   31
.extern L3
```

להלן טבלת הקידוד הבינארי המלא שמתקבל מקובץ המקור, כפי שנבנה במעבר הראשון והשני.

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000010000010010
0000102 0000103	LOOP: prn #48	Immediate value 48	001101000000000000000100 000000000000000110000100
0000104 0000105	lea W, r6	Address of extern label W	000100010001111000000100 000000000000000000000001
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	Address of label K	000000110110100000000100 000000000000010000101010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	Address of label END	001001000000100000010100 000000000000001111100010
0000112 0000113 0000114	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 000000000000010000101010 111111111111111111010100
0000115 0000116	bne &END	Distance to label END	001001000001000000010100 000000000000000001001100
0000117 0000118	dec W	Address of extern label W	000101000000100000100100 000000000000000000000001
0000119 0000120	jmp &LOOP	Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121 0000122 0000123	add L3, L3	Address of extern label L3 Address of extern label L3	000010010000100000001100 000000000000000000000001 000000000000000000000001
0000124	END: stop		001111000000000000000100
0000125	STR: .string “abcd”	Ascii code ‘a’	000000000000000001100001
0000126		Ascii code ‘b’	000000000000000001100010
0000127		Ascii code ‘c’	000000000000000001100011
0000128		Ascii code ‘d’	000000000000000001100100
0000129		Ascii code ‘\0’	000000000000000000000000
0000130 0000131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000132	.data -100	Integer -100	111111111111111110011100
0000133	K: .data 31	Integer 31	000000000000000000011111

טבלת הסמלים הסופית בגמר המעבר השני היא:

סמל	ערך (בבסיס עשרוני)	איפיון הסמל
W	0	external
MAIN	100	code, entry
LOOP	102	code
END	124	code
STR	125	data
LIST	130	data, entry
K	133	data
L3	0	external

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ **ps.ob**:

```
     25 9
0000100 0b680c
0000101 000412
0000102 340004
0000103 000184
0000104 111e04
0000105 000001
0000106 141e1c
0000107 036804
0000108 00042a
0000109 0b3c14
0000110 240814
0000111 0003e2
0000112 050004
0000113 00042a
0000114 ffffd4
0000115 241014
0000116 00004c
0000117 140824
0000118 000001
0000119 24100c
0000120 ffff7c
0000121 09080c
0000122 000001
0000123 000001
0000124 3C0004
0000125 000061
0000126 000062
0000127 000063
0000128 000064
0000129 000000
0000130 000006
0000131 fffff7
0000132 ffff9c
0000133 00001f
```

הקובץ **ps.ext**:

```
W 0000105
W 0000118
L3 0000122
L3 0000123
```

הקובץ **ps.ent**:

```
MAIN 0000100
LIST 0000130
```

לתשומת לב: אם בקובץ המקור אין הנחיות extern. אזי לא ייווצר קובץ ext. בדומה, אם אין בקובץ המקור הנחיות entry., לא ייווצר קובץ ent. **אין ליצור קובץ ext או ent שנשאר ריק**.
הערה: אין חשיבות לסדר השורות בקבצים מסוג ent. או ext. כל שורה עומדת בפני עצמה.

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן אורך התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם כדי להקל במימוש האסמבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המאקרו), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).

- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם: prog.ob, prog.ext, prog.ent

- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. כלומר, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתכנית האסמבלר כארגומנטים בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.

- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון: מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודי הפעולה, שיטות המיעון החוקיות לכל פעולה, וכד').

- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.

- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).

- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

בשאלות ניתן לפנות לקבוצת הדיון באתר הקורס, ואל המנחים בשעות הקבלה.

להזכירכם, מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר הרבה שאלות בנושא חומר הלימוד והממ"נים, והתשובות יכולות להועיל לכולם.

לתשומת לבכם:

- על המטלה להיות מקורית לגמרי: **אין** להיעזר בספריות חיצוניות מלבד הספריות הסטנדרטיות, וכמובן לא בקוד שמצאתם ברשת או קיבלתם בכל דרך. **אין** לשתף ברשת קוד ללא סיסמה. אלו הן **עבירות משמעת.**
- לא תינתן דחיה בהגשת הממ"ן, פרט למקרים חריגים במיוחד. במקרים אלו יש לבקש ולקבל אישור **מראש ממנחה הקבוצה**.

ב ה צ ל ח ה !