ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – אביב 2025ב

כתבה: מיכל אבימור

מרץ 2025 – סמסטר אביב – 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 (20465 / ב2025)

שבוע לימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח ממ"ן (למנחה)
1	14.03.2025-09.03.2025		מפגש ראשון	
2	21.03.2025-16.03.2025			
3	28.03.2025-23.03.2025		מפגש שני	
4	04.04.2025-30.03.2025			
5	11.04.2025-06.04.2025		מפגש שלישי	
6	18.04.2025-13.04.2025 (א-ו פסח)			ממ"ן 11 13.04.2025
7	25.04.2025-20.04.2025 (ד יום הזכרון לשואה)		מפגש רביעי	
8	02.05.2025-27.04.2025 (ד יום הזיכרון, ה יום העצמאות)			
9	09.05.2025-04.05.2025		מפגש חמישי	ממ"ן 12 04.05.2025

** 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)
10	16.05.2025-11.05.2025 (ו ל"ג בעומר)			
11	23.05.2025-18.05.2025		מפגש שישי	
12	30.05.2025-25.05.2025			ממ"ן 22 25.05.2025
13	06.06.2025-01.06.2025 (ב שבועות)		מפגש שביעי	
14	13.06.2025-08.06.2025		מפגש שמיני	ממ"ן 23 08.06.2025
15	20.06.2025-15.06.2025			ממ"ן 14** 12.08.2025

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: 2025ב' **מועד אחרון להגשה:** 13.04.2025

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 palindrome.c) (50 נקודות)

עליכם לכתוב תכנית המכילה פונקציה:

```
int palindrome(char s[ ]);
```

המקבלת כפרמטר מחרוזת תווים s, ומחזירה 1 אם s פלינדרום, 0 אחרת.

תזכורת: מילה היא פלינדרום אם כאשר קוראים את המילה משמאל לימין או מימין לשמאל, מתקבלת אותה המילה. לדוגמה: המילים אמא, אבא הן פלינדרומים. המילים סבא, סבתא אינן פלינדרומים.

על הפונקציה להתעלם מרווחים וטאבים (tab) במחרוזת s. כמו כן, יש להתעלם מהתו \0 המסיים את המחרוזת. אותיות קטנות וגדולות נחשבות שונות זו מזו.

לדוגמה:

עבור הפרמטר:	a b c b a	נקבל 1,
עבור הפרמטר:	never odd or even	נקבל 1,
וכן עבור הפרמטר:	abba	נקבל 1.
לעומת זאת עבור:	Abba	נקבל 0,
וכן עבור:	a bc de	נקבל 0.

בנוסף, עליכם לכתוב תכנית ראשית (הפונקציה main), שתבצע קלט של מחרוזת מהמשתמש, תפעיל את הפונקציה על מחרוזת הקלט, ותדפיס באופן נאה וידידותי את התוצאה.

המחרוזת מועברת בשורת קלט אחת. מותר להניח שהאורך המקסימלי של שורת הקלט הוא 80 תווים (לא כולל התו \n, שאינו נחשב חלק מהמחרוזת).

הקלט לתכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט, והשתמשו בהם שוב ושוב לדיבוג התכנית. הפלט הוא stdout ויכול להיות המסך או קובץ.

על התכנית **להדפיס הודעת בקשה** ידידותית לקלט. כמו כן יש **להדפיס באופן נאה את המחרוזת כפי שנקלטה**, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את פעולת התכנית על מגוון נתוני קלט. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות**, יחד עם כל קבצי הקלט.

שאלה 2 (תכנית ראשית בקובץ count_zero_bits.c) (50 נקודות)

עליכם לכתוב פונקציה בשם count_zero_bits, אשר סופרת כמה סיביות עם **ערך 0** יש בייצוג הבינרי של מספר שלם מטיפוס unsigned int, אותו היא מקבלת כפרמטר. הפונקציה מחזירה את מספר הסיביות עם ערך 0.

על הפונקציה count_zero_bits להיות ניידת (portable), כלומר עליה להיות בלתי תלויה ברוחב הייצוג של ערכים בסביבת ריצה מסוימת (מעבד, או מערכת הפעלה או מהדר -compiler - מסוימים).

כמו כן, עליכם לכתוב תכנית ראשית (הפונקציה main), שתקלוט מהמשתמש מספר שלם <u>בבסיס עשרוני</u>, ותקרא לפונקציה count_zero_bits עם מספר זה כפרמטר. אחרי החזרה מהפונקציה, התכנית הראשית תדפיס באופן נאה את המספר שנקלט (בבסיס עשרוני) ואת מספר הסיביות עם ערך 0 שבו.

הקלט לתוכנית הוא מהמקלדת. לתשומת לב: כל הרצה של התכנית מטפלת במספר יחיד בקלט.

על התוכנית **להדפיס הודעת בקשה ידידותית לקלט** המפרטת מה על המשתמש להקליד. הניחו כי הקלט תקין, כלומר מספר שלם עשרוני שאינו חורג מהערך המקסימלי/מינימלי האפשרי בטיפוס unsigned int בסביבת הריצה הנתונה. אין צורך לבדוק שגיאות בקלט.

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: 2025ב' **מועד אחרון להגשה:** 04.05.2025

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. 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.

<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 (תכנית ראשית בקובץ magic.c)

ריבוע קסם בסיסי הוא מטריצה בגודל N x N המכילה ערכים שלמים מ- 1 עד N^2 (כולל), כך שסכום N האיברים בכל שורה, בכל עמודה, ובכל אלכסון הוא זהה (משותף).

<u>דוגמה</u>: להלן ריבוע קסם בסיסי עבור N = 5, כאשר הסכום המשותף הוא 65.

15	**8**	**1**	**24**	**17**
16	**14**	**7**	**5**	**23**
22	**20**	**13**	**6**	**4**
3	**21**	**19**	**12**	**10**
9	**2**	**25**	**18**	**11**

עליכם לכתוב תכנית, המטפלת במטריצה בגודל N x N עם איברים מטיפוס int. המימד N מוגדר בתכנית באמצעות הנחיית #define. כמובן, ניתן לשנות את N מדי פעם ולקמפל מחדש.

התכנית קוראת מהקלט הסטנדרטי N^2 ערכים מטיפוס int בזה אחר זה. אלו הם איברי המטריצה, המועברים לפי סדר השורות, ומשמאל לימין בכל שורה. הערכים מופרדים זה מזה בקלט באמצעות תווים לבנים בלבד (אחד או יותר).

התכנית מדפיסה לפלט הסטנדרטי את המטריצה בתצוגה דו-מימדית נאה (ראו הדוגמה לעיל), וכן הודעה האם המטריצה מהווה ריבוע קסם בסיסי, או לא.

<u>אין להניח</u> שהקלט תקין. על התוכנית להפסיק את עבודתה, ולהדפיס הודעת שגיאה במקרים הבאים: אין מספיק ערכים בקלט; יש יותר מדי ערכים בקלט; יש בקלט ערך שאינו מטיפוס int.

לתשומת לב: ערך מטיפוס int שחורג מהתחום $1–N^2$, או יותר ממופע אחד של אותו ערך, <u>אינם נחשבים כשגיאת קלט</u> (אם כי כמובן המטריצה לא תהיה ריבוע קסם בסיסי).

נדרש לבנות את התכנית באופן מודולרי ולחלק את העבודה לפונקציות בצורה מושכלת, כלומר להשתמש בפונקציות נפרדות למשימות שונות, כגון קלט, בדיקות, פלט, וכו'.

<u>הנחיות והערות נוספות:</u>

- בתחילת הריצה, על התוכנית **להדפיס הודעת בקשה ידידותית לקלט,** המפרטת מה על המשתמש להקליד, לרבות מהו מספר הערכים הנדרש (לפי המימד N הקבוע בקוד).
- <u>אין להגביל</u> את ארגון הקלט באופן כלשהו. כלומר, המשתמש יכול להעביר כרצונו כל כמות ערכים בכל שורת קלט. למשל, כל שורה של המטריצה בשורת קלט נפרדת, או כל המטריצה בשורת קלט אחת, או כל ערך בשורה נפרדת, 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.
- רמז: ניתן לקבוע שאין מספיק ערכים בקלט אם מזהים בזרם הקלט מצב EOF (סוף הקלט) לפני שנקלטו N^2 ערכים. אפשר לדמות מצב EOF במקלדת באמצעות הקלדה של צרוף המקשים ctrl+d באובונטו, או ctrl+z בחלונות.
- לבניית ריבוע קסם בכל גודל אפשר להיעזר באתר: www.dcode.fr/magic-square

חובה לצרף להגשה מספר הרצות דוגמה. יש להציג מטריצות בכמה גדלים שונים, כאשר חלקן מהוות ריבוע קסם בסיסי וחלקן לא. כמו כן, יש להדגים את הטיפול בשגיאות מגוונות בקלט. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם אותם.**

<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: 2025ב' **מועד אחרון להגשה:** 25.05.2025

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.

<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 (בקבצים עיקריים mymat.c, mymat.h, ותכנית ראשית בקובץ mainmat.c)

עליכם לכתוב תכנית שפועלת כ"מחשב כיס" אינטראקטיבי לביצוע פעולות חשבוניות על מטריצות.

<u>תזכורת</u>:

להלן כמה פעולות חשבוניות בסיסיות על מטריצות.

חיבור מטריצות.
דוגמה:

```
1  2  3  4        2  2  2  0        3  4  5  4
1  2  3  4   +    2  2  2  0   =    3  4  5  4
1  2  3  4        2  2  2  0        3  4  5  4
1  2  3  4        2  2  2  0        3  4  5  4
```

חיסור מטריצות.
דוגמה:

```
1  1  1  1        0.5  0   1   0        0.5  1   0    1
2  2  2  2   -    2    2   0   0   =    0    0   2    2
1  1  1  1        1    1   0   1        0    0   1    0
2  2  2  2        1    0   2   3        1    2   0   -1
```

כפל מטריצות.
דוגמה:

```
1  1  1  1        1  0  1  0        5   3   3   4
2  2  2  2   *    2  2  0  0   =   10   6   6   8
1  1  1  1        1  1  0  1        5   3   3   4
2  2  2  2        1  0  2  3       10   6   6   8
```

כפל מטריצה בסקלר.
דוגמה:

```
1  0  1  0                  0.5    0     0.5    0
2  2  0  0   *    0.5   =   1      1     0      0
1  1  0  1                  0.5    0.5   0      0.5
1  0  2  3                  0.5    0     1      1.5
```

שחלוף (transposition) של מטריצה.
דוגמה:

```
 1  1  1  1          1   2  -1   2
 2  2  2  2     →    1   2   1   2
-1  1  1  1          1   2   1   2
 2  2  2  2          1   2   1   2
```

<u>משימות התכנית:</u>

עליכם לכתוב תכנית הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן.
הפקודות עוסקות בפעולות חשבוניות על מטריצות (על פי התזכורת לעיל).

עליכם להגדיר, תוך שימוש ב- typedef, את הטיפוס mat אשר מחזיק מטריצה בגודל 4 על 4.
איברי המטריצה הם מספרים ממשיים. מבנה הנתונים שהגדרתם צריך להיות חסכוני מבחינת ניצול הזיכרון, ויעיל מבחינת הגישה אליו.

בנוסף, עליכם להגדיר בתכנית הראשית (בפונקציה main) שישה משתנים מטיפוס mat, בשמות:
MAT_A, MAT_B, MAT_C, MAT_D, MAT_E, MAT_F

בתחילת ריצת התכנית , יש לאתחל את כל המטריצות עם אפסים בכל האיברים.

כעת, עליכם לבצע פעולות חשבוניות על מטריצות. כל פעולה תופעל באמצעות פקודה שמועברת מהמשתמש בקלט לתכנית. בפקודות שמפורטות להלן, כל אופרנד שהוא שם של מטריצה יהיה אחד מששת המשתנים שהוגדרו לעיל. כיוון הקריאה של נוסח הפקודה הוא משמאל לימין.

<u>מבנה הפקודות המשמשות כקלט לתכנית:</u>

1. הצבת ערכים במטריצה:

רשימת ערכים ממשיים מופרדים בפסיקים ,שם-מטריצה read_mat

הפקודה מציבה את הערכים שברשימה לתוך המטריצה ששמה ניתן בפקודה, לפי סדר השורות. אם ברשימה יש פחות מ-16 ערכים, האיברים שלא נתקבל עבורם ערך יכילו אפסים. אם יש יותר מ-16 ערכים, התכנית תתעלם מהערכים העודפים. חובה שיהיה ברשימה לפחות ערך אחד.

הערכים בקלט הם מספרים ממשיים בבסיס עשרוני.

לדוגמה, הפקודה: read_mat MAT_A, 5, 6.253, -7, -200.5, 23
תציב בתא [0,0] במטריצה MAT_A את הערך 5, בתא [0,1] את הערך 6.253, בתא[0,2] את הערך 7-. בתא [0,3] את הערך 200.5-, ובתא [1,0] את הערך 23. ביתר תאי המטריצה MAT_A יוצב הערך 0.

2. הדפסת מטריצה:

שם מטריצה print_mat

הפקודה מדפיסה את תוכן המטריצה ששמה ניתן בפקודה, בתצוגה דו-מימדית נאה. הערכים יודפסו בבסיס עשרוני.

יש להקפיד בהדפסה על עימוד נאה ומיושר של אברי המריצה. זכרו שמדובר במספרים ממשיים. מותר להסתפק בהדפסה עם דיוק של 2 ספרות מימין לנקודה, וכן ברוחב שדה של 7 תווים לכל המספר (כולל נקודה-עשרונית וסימן מינוס לפי הצורך). במידה והחלק השלם של מספר דורש רוחב שדה גדול יותר, מותרת סטיה מהעימוד המיושר בשורה זו. מומלץ להשתמש ביכולות של הפונקציה printf לשלוט בפורמט של השדה המודפס.

לדוגמה: הפקודה print_mat MAT_A (לאחר ביצוע דוגמת הפקודה read_mat מהסעיף הקודם) תבצע הדפסה בסגנון הבא (או דומה לו):

```
 5.00     6.25    -7.00  -200.50
23.00     0.00     0.00     0.00
 0.00     0.00     0.00     0.00
 0.00     0.00     0.00     0.00
```

3. חיבור מטריצות:

שם-מטריצה-ג' , שם-מטריצה-ב' , שם-מטריצה-א' add_mat

הפקודה מחברת את מטריצה א' ומטריצה ב' ומאכסנת את התוצאה במטריצה ג'.

4. חיסור מטריצות:

שם-מטריצה-ג' , שם-מטריצה-ב' , שם-מטריצה-א' sub_mat

הפקודה מחסרת את מטריצה ב' ממטריצה א' ומאכסנת את התוצאה במטריצה ג'.

5. כפל ממטריצות:

שם-מטריצה-ג' , שם-מטריצה-ב' , שם-מטריצה-א' mul_mat

הפקודה מכפילה את מטריצה א' במטריצה ב' ומאכסנת את התוצאה במטריצה ג'.

6. כפל מטריצה בסקלר:

שם-מטריצה ב' , ערך-ממשי , שם-מטריצה-א' mul_scalar

הפקודה מכפילה את מטריצה א' בערך ממשי (הפרמטר השני) ומאכסנת את התוצאה במטריצה ב'. הערך הממשי נתון בבסיס עשרוני.

7. שחלוף מטריצה:

trans_mat שם מטריצה ב' , שם-מטריצה-א'

הפקודה מבצעת שחלוף (transpose) של מטריצה א' ומאכסנת את התוצאה במטריצה ב'.

8. **סיום התכנית:**

stop

הפקודה גורמת לסיום התכנית.

<u>לתשומת לב:</u> אותו שם מטריצה יכול לשמש ביותר מארגומנט אחד באותה הפקודה. מימוש הפעולות החשבוניות על מטריצות צריך להתחשב בכך (לא לדרוס נתונים תוך כדי החישוב).
לדוגמה, הפקודות שלהלן תקינות ומוגדרות היטב:

mul_mat MAT_A, MAT_B, MAT_A

trans_mat MAT_C, MAT_C

<u>המבנה התחבירי של הקלט</u>:

- כל פקודה תופיע בשלמותה בשורת קלט יחידה, כולל כל הארגומנטים. מותרות גם שורות ריקות (שורות המכילות רק תווים לבנים).
- שם הפקודה מופרד מהארגומנט הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).
- בין כל שני ארגומנטים 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 בתחילת השורה לפני שם הפקודה, וגם בסוף השורה (אחרי הארגומנט האחרון).
- אסור שיהיו תווים מיותרים (תווי זבל) בסוף השורה (למעט תווים לבנים).
- שמות הפקודות יופיעו באותיות קטנות בלבד, ושמות המשתנים באותיות גדולות בלבד.

<u>אופן פעולת התכנית</u>:

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

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

יש לטפל גם במצב של EOF (גמר הקלט). סיום התכנית שלא באמצעות פקודת stop מפורשת בקלט אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ באמצעות redirection), ויש להדפיס על כך הודעת שגיאה ורק אז לעצור.
לתשומת לב: השורה האחרונה בקובץ קלט <u>אינה חייבת</u> להסתיים בתו "שורה חדשה".

<u>להלן דוגמאות של קלט שגוי</u>:

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

1. לפקודה:

read_mat MAT_G, 3.2, 8

יש להגיב בהודעה כגון:

Undefined matrix name

2. לפקודה:

read_mat mat_a, 3.2, -5.3

יש להגיב בהודעה כגון:

Undefined matrix name

3. לפקודה:

do_it MAT_A, MAT_B, MAT_C

יש להגיב בהודעה כגון:

Undefined command name

4. לפקודה:

Add_Mat MAT_A, MAT_B, MAT_C

יש להגיב בהודעה כגון:

Undefined command name

5. לפקודה:

read_mat MAT_A, abc, 567

יש להגיב בהודעה כגון:

Argument is not a real number

6. לפקודה:

read_mat MAT_A, 3, -4.2, 6,

יש להגיב בהודעה כגון:

Extraneous text after end of command

7. לפקודה:

read_mat MAT_A

יש להגיב בהודעה כגון:

Missing argument

8. לפקודה:

mul_mat MAT_B, MAT_C

יש להגיב בהודעה כגון:

Missing argument

9. לפקודה:

trans_mat MAT_B, MAT_C, MAT_D

יש להגיב בהודעה כגון:

Extraneous text after end of command

10. לפקודה:

print_mat, MAT_A

יש להגיב בהודעה כגון:

Illegal comma

11. לפקודה:

trans_mat MAT_A MAT_B

יש להגיב בהודעה כגון:

Missing comma

12. לפקודה:

sub_mat MAT_A, , MAT_B, MAT_C

יש להגיב בהודעה כגון:

Multiple consecutive commas

13. לפקודה:

mul_scalar MAT_A, MAT_B, MAT_C

יש להגיב בהודעה כגון:

Argument is not a scalar

14. לפקודה:

stop now

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 ללא טיפול בשגיאות בקלט.

```
print_mat MAT_A
print_mat MAT_B
print_mat MAT_C
read_mat MAT_A, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 6, 6, 6, 6, 6
read_mat MAT_B, 1, 2.3456, -7.89
read_mat MAT_C, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
print_mat MAT_A
print_mat MAT_B
print_mat MAT_C
add_mat MAT_A, MAT_B, MAT_D
print_mat MAT_D
sub_mat MAT_B, MAT_A, MAT_E
print_mat MAT_E
mul_mat MAT_B, MAT_C, MAT_F
print_mat MAT_F
mul_scalar MAT_A, 12.5, MAT_A
print_mat MAT_A
trans_mat MAT_C, MAT_C
print_mat MAT_C
read_mat MAT_B,      0
print_mat      MAT_B
mul_mat MAT_A,MAT_A,MAT_A
print_mat MAT_A
stop
```

<u>ארגון קוד התכנית</u>:
יש לחלק את התכנית למספר קבצי מקור: mainmat.c ,mymat.h ,mymat.c

- בקובץ mymat.c יש לרכז את הפונקציות החישוביות על מטריצות. לא יבוצע כל קלט/פלט בקובץ זה.
- בקובץ mainmat.c תהיה הפונקציה main, וכן כל פעילויות האינטראקציה עם המשתמש, וניתוח הפקודות (לרבות הודעות שגיאה). כמו כן, בפונקציה main יוגדרו ששת המשתנים מטיפוס mat.
- בקובץ mymat.h תהיה הגדרת טיפוס הנתונים mat, וכן ההצהרות (אב-טיפוס) של הפונקציות הממומשות בקובץ mymat.c. יש לכלול (#include) את הקובץ mymat.h בקבצי המקור האחרים.
- ניתן לבנות קבצי מקור נוספים (למשל: קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').

הקלט לתכנית הוא מ-stdin, ויכול להגיע מהמקלדת או מקובץ (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התכנית. בכל קובץ קלט תהיה סדרה של פקודות מגוונות על מטריצות.

על התכנית **להדפיס הודעת בקשה ידידותית לקלט** עבור כל שורת קלט (כל פקודה). כמו כן, יש **להדפיס באופן יזום מתוך התכנית את השורה כפי שנקלטה**, וזאת <u>לפני</u> הניתוח של הפקודה.

באופן זה, שורת הקלט תוצג גם כאשר הקלט מגיע מקובץ (כידוע, נתונים הנקראים מקובץ אינם מוצגים במסך בזמן הקלט).

חובה לצרף להגשה <u>הרצות דוגמה</u> (אחת או יותר), המדגימות את השימוש בכל סוגי הפעולות ובכל ששת המטריצות, וכן את הטיפול בכל מגוון השגיאות בקלט.
<u>רמז</u>: מומלץ להכניס פקודת הדפסה של מטריצת התוצאה אחרי כל פעולה, כדי להראות שהתוצאה אכן נכונה (ראו לעיל הדוגמה של סדרת פקודות תקינות).
יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות. יש להגיש גם קבצי קלט.

<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: 2025ב' **מועד אחרון להגשה:** 08.06.2025

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: -pedantic -ansi -Wall . 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.

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 (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. קובץ הקלט הסטנדרטי stdin הוא תמיד המקלדת, קובץ הפלט הסטנדרטי stdout הוא תמיד המסך, וקובץ השגיאות הסטנדרטי stderr הוא תמיד המסך.

ב. בגרסת ANSI של שפת C ניתן לשנות את תחום האינדקסים בהגדרה של מערך. ברירת המחדל היא אינדקסים החל מ- 0, אך ניתן לשנות זאת לאינדקסים החל מ- 1.

לתשומת לב:

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

שאלה 2 (90 נקודות) (תוכנית ראשית בקובץ numbers.c)

עליכם לכתוב תכנית המקבלת כארגומנטים בשורת הפקודה (**command line arguments**) 0, 1 או 2 שמות של קבצים.

אם מופיעים שני שמות של קבצים, הקובץ הראשון הוא קובץ הקלט והקובץ השני הוא קובץ הפלט.
אם יש שם קובץ אחד, יהיה זה שם קובץ הקלט, והפלט ישלח לקובץ הפלט הסטנדרטי (**stdout**).
אם אין אף שם של קובץ, הקלט יילקח מקובץ הקלט הסטנדרטי (**stdin**), והפלט יישלח לקובץ הפלט הסטנדרטי (**stdout**).

אם יש יותר משני ארגומנטים בשורת הפקודה, על התוכנית להדפיס הודעת שגיאה מפורטת ולהפסיק את עבודתה. אם קובץ הקלט לא קיים, או אם קובץ הקלט או קובץ הפלט אינם ניתנים לפתיחה, על התוכנית להדפיס הודעת שגיאה מפורטת ולהפסיק את עבודתה.

את הודעות השגיאה יש לשלוח לקובץ השגיאות הסטנדרטי (**stderr**).

הקלט מכיל סדרת מספרים עשרוניים שלמים, בתחום 99-0, מופרדים זה מזה בתווים לבנים (אחד או יותר). על התוכנית לקרוא את המספרים מהקלט, <u>להמיר כל מספר</u> למילים (באנגלית) <u>ולהדפיס</u> מילים אלה לפלט. לכל מספר, הפלט יופיע בשורה נפרדת.

לדוגמה, עבור קובץ קלט המכיל:

75 56 32 5 12 54 0 99 17

קובץ הפלט יכיל:

seventy five
fifty six
thirty two
five
twelve
fifty four
zero
ninety nine
seventeen

כמות המספרים בקלט אינה מוגבלת. הקלט מסתיים כאשר מתגלה EOF.
הניחו שהקלט תקין. אין צורך לבדוק שגיאות בקלט.

<u>הערה</u>: על המילים בפלט להיות בפורמט דומה למופיע בדוגמה לעיל (אותיות קטנות, רווח יחיד).

חובה לצרף להגשה מספר הרצות בדיקה, לרבות הרצות שנותנות הודעת שגיאה. יש להדגים את פעולת התוכנית על ערכים מגוונים של מספרים, וכן את כל הקומבינציות של ארגומנטים לתוכנית.
לכל הרצה, יש להגיש את קובץ הקלט וקובץ הפלט (ככל שקיימים) , וכן את תדפיס המסך.

<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: 2025ב' | מועד אחרון להגשה: 12.08.2025

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.

יש להגיש לפחות 3 קבצי קלט ו-3 קבצי פלט ותדפיסי מסך המייצגים מצבים שונים (תקינים ושגיאות)

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 10 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 9. 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 256 תאים, בכתובות 0-255 (בבסיס עשרוני), וכל תא הוא בגודל של 10 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).
במחשב זה 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 חמש 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:

9 8 7 6	5 4	3 2	1 0
opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד כל מילה בקוד המכונה ייעשה בבסיס 4 "ייחודי" המוגדר כדלקמן: ארבע הספרות הן: a,b,c,d (כאשר a שקול ל-0, b ל-1, c ל-2, d ל-3). קידוד של מילה בגודל 10 סיביות בבסיס 4 מורכב מחמש ספרות (עם ספרות a מובילות לפי הצורך).

סיביות 6-9 במילה הראשונה של הפקודה מהוות את קוד ההוראה (opcode). בשפה שלנו יש 16 קודי הוראה והם:

הקוד בבסיס דצימלי (10)	פעולה
0	mov
1	cmp
2	add
3	sub
4	not
5	clr
6	lea
7	inc
8	dec
9	jmp
10	bne
11	red
12	prn
13	jsr
14	rts
15	stop

ההוראות נכתבות תמיד באותיות קטנות. פרוט משמעות ההוראות יבוא בהמשך.

סיביות 0-1 (A,R,E)

סיביות אלה מראות את סוג הקידוד, האם הוא מוחלט (Absolute) , חיצוני (External) או מצריך מיקום מחדש (Relocatable)

ערך של 00 משמעו שהקידוד הוא מוחלט.
ערך של 01 משמעו שהקידוד הוא חיצוני.
ערך של 10 משמעו שהקידוד מצריך מיקום מחדש.

סיביות אלה מתווספות רק לקידודים של הוראות (לא של נתונים), והן מתווספות גם לכל המילים הנוספות שיש לקידודים אלה.

סיביות 2-3 מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand).

סיביות 4-5 מקודדות את מספרה של שיטת המיעון של אופרנד המקור (source operand) .

בשפה שלנו קיימות ארבע שיטות מיעון, שמספרן הוא בין 0 ל- 3.

השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בהוראה, לכל היותר שתי מילים נוספות לכל אופרנד. אם שיטת המיעון של רק אחד משני האופרנדים דורשת מילות מידע נוספות, אזי מילות המידע הנוספות מתייחסות לאופרנד זה. אם שיטות המיעון של שני האופרנדים דורשות מילות-מידע נוספות, אזי מילות-המידע הנוספות הראשונות מתייחסות לאופרנד המקור ומילות-המידע הנוספות האחרונות מתייחסות לאופרנד היעד.

ארבע שיטות המיעון הקיימות במכונה שלנו הן:

מספר	שיטת המיעון	תוכן מילת המידע הנוספת	אופן כתבת האופרנד	דוגמה
0	מיעון מידי	המילה הנוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 8 סיביות, אליהם מתווספות זוג סיביות של שדה A,R,E	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני	mov #-1,r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מיידי. הפקודה כותבת את הערך 1- לתוך אוגר r2
1	מיעון ישיר	המילה הנוספת של ההוראה מכילה מען של מילה בזיכרון. מילה זו בזיכרון הינה האופרנד. המען מיוצג ב- 8 סיביות אליהן מתווספות זוג סיביות של שדה A,R,E	האופרנד הינו <u>תווית</u> שהוצהרה או תוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בקובץ המקור (בהנחיית 'data.'או 'string.' או 'mat.', או בתחילת הוראה של התוכנית), או על ידי אופרנד של הנחית 'extern.'	dec x בדוגמה זו, תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x) מוקטן ב-1.
2	מיעון גישה למטריצה	בשיטת מיעון זו משתתפים שני אוגרים ותווית. בזמן ביצוע ההוראה, המעבד יפנה למטריצה (המצוינת ע"י התוית), לתא (בגודל מילה) הנמצא באינדקסים (שורה ועמודה) המצוינים ע"י האוגרים. בשיטת מיעון זו יש 2 מילות מידע נוספות בקוד ההוראה. המילה הנוספת הראשונה היא כתובת התחלת המטריצה (מצוין ע"י התוית). והמילה הנוספת השניה תכיל את האינדקסים באופן הבא: ארבע הסיביות 9-6 מקודדים את מספר האוגר המכיל את אינדקס השורה , וארבע הסיביות 5-2 מקודדים את מספר האוגר המכיל את אינקס העמודה. האינדקסים מתחילים ב-0. לייצוג זה מתווספות זוג סיביות של שדה A,R,E.	האופרנד מורכב משם של תוית המציינת מטריצה , ולאחריה שורה ועמודה במטריצה המצוינים ע"י אוגרים בלבד, ורשומים כל אחד בסוגריים מרובעות.	add #4, a[r2][r5] בדוגמה זו, הפקודה מוסיפה את הערך 4 לתא במטריצה שנמצאת בתווית a. התא הוא בשורה המצוינת ע"י תוכן האוגר r2 ובעמודה המצוינת ע"י תוכן האוגר r5
3	מיעון אוגר ישיר	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בארבע הסיביות 5-2 את מספרו של האוגר. אם האוגר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 9-6 את מספרו של האוגר. אם בפקודה יש שני אופרנדים ושניהם אוגרים, הם יחלקו מילה נוספת אחת משותפת. כאשר הסיביות 5-2 הן עבור אוגר היעד, והסיביות	האופרנד הינו שם של אוגר.	mov r1,r2 בדוגמה זו, אופרנד המקור הוא האוגרr1, ואופרנד היעד הוא האוגר r2. הפקודה מעתיקה את תוכן אוגר r1 לתוך אוגר r2. בדוגמה זו שני האופרנדים יקודדו למילה משותפת.

		6-9 הן עבור אוגר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0.		

<u>הערה:</u> מותר להתייחס לתווית עוד לפני שמצהירים עליה, בתנאי שהיא אכן מוצהרת במקום כלשהו בקובץ.

<u>מפרט הוראות המכונה:</u>

בתיאור הוראות המכונה נשתמש במונח **PC** (קיצור של "**Program Counter**").
זהו רגיסטר פנימי של המעבד (<u>לא</u> 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	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	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	3	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	6	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 4-5) 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	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	איפוס תוכן האופרנד	clr r2	הרגיסטר r2 מקבל את הערך 0.
not	4	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט ברגיסטר r2 מתהפך.
inc	7	הגדלת תוכן האופרנד באחד.	inc r2	תוכן הרגיסטר r2 מוגדל ב-1.
dec	8	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב- 1.
jmp	9	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את ערך אופרנד היעד.	jmp Line	PC ← Line
bne	10	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 יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הוא 0, אזי PC ← Line
jsr	13	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	11	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לרגיסטר r1.
prn	12	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 5-2) 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). אם השורה ארוכה מ-80 תוים, יש לדווח על שגיאה.

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. 'mat.'

משפט הנחיה זה מקצה מטריצה. למשפט הנחיה 'mat.' המבנה הבא:

```
MAT8:  .mat [2][3]
MAT5:  .mat [2][2] 4,-5,7,9
```

בדוגמא הראשונה מקצים מטריצה בשם MAT8 שגודלה 2 שורות ו-3 עמודות והיא אינה מאותחלת בערכים (אבל כן צורכת מקום בתמונת הנתונים).

בדוגמא השניה מקצים מטריצה בשם MAT5 בגודל 2 שורות ו-2 עמודות, המאותחלת לערכים המפורטים. הערכים לאתחול רשומים משמאל לימין לפי סדר השורות.
מטריצה תכיל רק מספרים שלמים.

4. 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).

5. 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
```

הערה: לא ניתן להגדיר באותו הקובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO). במקרה כזה יש להודיע על שגיאה.

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

משפט הוראה:

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
```

אפיון השדות במשפטים של שפת האסמבלי

תווית:
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 30 תווים.

הגדרה של תווית 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 שונות זו מזו.

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, .mat' 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:     mov     M1[r2][r7],LENGTH
          add     r2,STR
LOOP:     jmp     END
          prn     #-5
          mcro  a_mc
          mov     M1[r3][r3],r3
          bne     LOOP
          mcroend
          sub     r1, r4
          inc     K

          a_mc
END:      stop
STR:      .string  "abcdef"
LENGTH:   .data    6,-9,15
K:        .data    22
M1:       .mat  [2][2]  1,2,3,4
```

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:     mov     M1[r2][r7],LENGTH
          add     r2,STR
LOOP:     jmp     END
          prn     #-5
          sub     r1, r4
          inc     K

          mov     M1[r3][r3],r3
          bne     LOOP
END:      stop
STR:      .string  "abcdef"
LENGTH:   .data    6,-9,15
K:        .data    22
M1:       .mat  [2][2]  1,2,3,4
```

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

אלגוריתם שלדי של קדם האסמבלר

נציג להלן אלגוריתם שלדי לתהליך קדם האסמבלר. <u>לתשומת לב</u>: 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 LOOP, 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:

<u>לתשומת לב</u>: המקפים המופיעים בקידוד הבינרי הם רק לצורך הדגשת ההפרדה בין השדות השונים בקידוד ונועדו לשם המחשה בלבד.

Label	Decimal Address	Base 4 Address	Instruction	Operands	Binary machine code
MAIN:	**0100**	1210	**mov**	M1[r2][r7],LENGTH	**0000-10-01-00**
	0101	1211		כתובת של M1	**10000101-10**
	0102	1212		קידוד אוגרי האינדקסים במטריצה	**0010-0111-00**
	0103	1213		כתובת של LENGTH	**10000001-10**
	0104	1220	**add**	r2, STR	**0010-11-01-00**
	0105	1221		קידוד מספר האוגר	**0010-0000-00**
	0106	1222		כתובת של STR	**01111010-10**
LOOP:	**0107**	1223	**jmp**	END	**1001-00-01-00**
	0108	1230		כתובת של END	**01111001-10**
	0109	1231	**prn**	#-5	**1100-00-00-00**

	0110	1232		המספר 5-	**11111011-00**
	0111 **0112**	1233 1300	**sub**	r1,r4 קידודי מספרי האוגרים	**0011-11-11-00** **0001-0100-00**
	0113 **0114**	1301 1302	**inc**	K כתובת של K	**0111-00-01-00** **10000100-10**
	0115 **0116** **0117** **0118**	1303 1310 1311 1312	**mov**	M1[r3][r3],r3 כתובת של M1 קידוד אוגרי האינדקסים במטריצה קידוד מספר האוגר של היעד	**0000-10-11-00** **10000101-10** **0011-0011-00** **0000-0011-00**
	0119 **0120**	1313 1320	**bne**	LOOP כתובת של LOOP	**1010-00-01-00** **01101011-10**
END:	**0121**	1321	**stop**		**1111-00-00-00**
STR:	**0122**	1322	**.string**	"abcdef"	**0001100001**
	0123	1323			**0001100010**
	0124	1330			**0001100011**
	0125	1331			**0001100100**
	0126	1332			**0001100101**
	0127	1333			**0001100110**
	0128	2000			**0000000000**
LENGTH:	**0129** **0130** **0131**	2001 2002 2003	**.data**	6,-9,15	**0000000110** **1111110111** **0000001111**
K:	**0132**	2010	**.data**	22	**0000010110**
M1	**0133** **0134** **0135** **0136**	2011 2012 2013 2020	**.mat**	[2][2] 1,2,3,4	**0000000001** **0000000010** **0000000011** **0000000100**

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), 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	107
END	121
STR	122
LENGTH	129
K	132
M1	133

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, .mat).

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 מייצר, 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 שיטות המיעון החוקיות, עבור אופרנד המקור ואופרנד היעד של ההוראות השונות הקיימות בשפה הנתונה:

Opcode	שם ההוראה	שיטות מיעון חוקיות עבור אופרנד המקור	שיטות מיעון חוקיות עבור אופרנד היעד
0	mov	0,1,2,3	1,2,3
1	cmp	0,1,2,3	0,1,2,3
2	add	0,1,2,3	1,2,3
3	sub	0,1,2,3	1,2,3
4	lea	1,2	1,2,3
5	clr	אין אופרנד מקור	1,2,3
6	not	אין אופרנד מקור	1,2,3
7	inc	אין אופרנד מקור	1,2,3
8	dec	אין אופרנד מקור	1,2,3
9	jmp	אין אופרנד מקור	1,2,3
10	bne	אין אופרנד מקור	1,2,3
11	jsr	אין אופרנד מקור	1,2,3
12	red	אין אופרנד מקור	1,2,3
13	prn	אין אופרנד מקור	0,1,2,3
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 0**.

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 0.
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. או mat.? 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 (ראו 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 mat. 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:     mov    M1[r2][r7],LENGTH
          add    r2,STR
LOOP:     jmp    END
          prn    #-5
          sub    r1, r4
          inc    K

          mov    M1[r3][r3],r3
          bne    LOOP
END:      stop
STR:      .string "abcdef"
LENGTH:   .data  6,-9,15
K:        .data  22
M1:       .mat  [2][2]  1,2,3,4
```

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). נניח שהקוד ייטען החל מהמען 100 (בבסיס 10).

Label	Decimal Address	Base 4 Address	instruction	Operands	Binary machine code
MAIN:	**0100**	1210	**mov**	M1[r2][r7],LENGTH	**0000-10-01-00**
	0101	1211		כתובת של M1	?
	0102	1212		קידוד אוגרי האינדקסים במטריצה	**0010-0111-00**
	0103	1213		כתובת של LENGTH	?
	0104	1220	**add**	r2, STR	**0010-11-01-00**
	0105	1221		קידוד מספר האוגר	**0010-0000-00**
	0106	1222		כתובת של STR	?
LOOP:	**0107**	1223	**jmp**	END	**1001-00-01-00**
	0108	1230		כתובת של END	?
	0109	1231	**prn**	#-5	**1100-00-00-00**
	0110	1232		המספר 5-	**11111011-00**
	0111	1233	**sub**	r1,r4	**0011-11-11-00**
	0112	1300		קידודי מספרי האוגרים	**0001-0100-00**
	0113	1301	**inc**	K	**0111-00-01-00**
	0114	1302		כתובת של K	**?**
	0115	1303	**mov**	M1[r3][r3],r3	**0000-10-11-00**
	0116	1310		כתובת של M1	**?**
	0117	1311		קידוד אוגרי האינדקסים במטריצה	**0011-0011-00**
	0118	1312		קידוד מספר האוגר של היעד	**0000-0011-00**
	0119	1313	**bne**	LOOP	**1010-00-01-00**
	0120	1320		כתובת של LOOP	**?**
END:	**0121**	1321	**stop**		**1111-00-00-00**
STR:	**0122**	1322	**.string**	“abcdef”	**0001100001**
	0123	1323			**0001100010**
	0124	1330			**0001100011**
	0125	1331			**0001100100**
	0126	1332			**0001100101**
	0127	1333			**0001100110**
	0128	2000			**0000000000**
LENGTH:	**0129**	2001	**.data**	6,-9,15	**0000000110**
	0130	2002			**1111110111**
	0131	2003			**0000001111**
K:	**0132**	2010	**.data**	22	**0000010110**
M1	**0133**	2011	**.mat**	[2][2] 1,2,3,4	**0000000001**
	0134	2012			**0000000010**
	0135	2013			**0000000011**
	0136	2020			**0000000100**

טבלת הסמלים אחרי המעבר ראשון היא:

סמל	ערך (בבסיס עשרוני)	איפיון הסמל
MAIN	100	code
LOOP	107	code
END	121	code
STR	122	data
LENGTH	129	data
K	132	data
M1	133	

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).

Label	Decimal Address	Base 4 Address	Command	Operands	Binary machine code
MAIN:	**0100**	1210	**mov**	M1[r2][r7],LENGTH	**0000-10-01-00**
	0101	1211		כתובת של M1	**10000101-10**
	0102	1212		קידוד אוגרי האינדקסים במטריצה	**0010-0111-00**
	0103	1213		כתובת של LENGTH	**10000001-10**
	0104	1220	**add**	r2, STR	**0010-11-01-00**
	0105	1221		קידוד מספר האוגר	**0010-0000-00**
	0106	1222		כתובת של STR	**01111010-10**
LOOP:	**0107**	1223	**jmp**	END	**1001-00-01-00**
	0108	1230		כתובת של END	**01111001-10**
	0109	1231	**prn**	#-5	**1100-00-00-00**
	0110	1232		המספר 5-	**11111011-00**
	0111	1233	**sub**	r1,r4	**0011-11-11-00**
	0112	1300		קידודי מספרי האוגרים	**0001-0100-00**
	0113	1301	**inc**	K	**0111-00-01-00**
	0114	1302		כתובת של K	**10000100-10**
	0115	1303	**mov**	M1[r3][r3],r3	**0000-10-11-00**
	0116	1310		כתובת של M1	**10000101-10**
	0117	1311		קידוד אוגרי האינדקסים במטריצה	**0011-0011-00**
	0118	1312		קידוד מספר האוגר של היעד	**0000-0011-00**
	0119	1313	**bne**	LOOP	**1010-00-01-00**
	0120	1320		כתובת של LOOP	**01101011-10**
END:	**0121**	1321	**stop**		**1111-00-00-00**
STR:	**0122**	1322	**.string**	"abcdef"	**0001100001**
	0123	1323			**0001100010**
	0124	1330			**0001100011**
	0125	1331			**0001100100**
	0126	1332			**0001100101**
	0127	1333			**0001100110**
	0128	2000			**0000000000**
LENGTH:	**0129**	2001	**.data**	6,-9,15	**0000000110**
	0130	2002			**1111110111**
	0131	2003			**0000001111**
K:	**0132**	2010	**.data**	22	**0000010110**
M1	**0133**	2011	**.mat**	[2][2] 1,2,3,4	**0000000001**
	0134	2012			**0000000010**
	0135	2013			**0000000011**
	0136	2020			**0000000100**

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. 'mat.'
האסמבלר קורא את רשימת המספרים של איתחול המטריצה (אם ישנו), מכניס כל מספר שנקרא אל מערך הנתונים, ומקדם את מצביע הנתונים באחד עבור כל מספר שהוכנס. אם המטריצה אינה מאותחלת בערכים, יוקצו תאים במערך בהתאם לגודל המטריצה, והם יאותחלו ל-0.

הטיפול בתווית המופיעה בשורה, זהה לטיפול הנעשה בהנחיה 'data.'.

III . '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.'.

IV. '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.

V. '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).

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט. השורה הראשונה מכילה שני מספרים: אורך כולל של קטע ההוראות (במילות זיכרון) ואורך כולל של קטע הנתונים (במילות זיכרון). השורות הבאות מתארות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. כל המספרים בקובץ object הם בבסיס 4 "ייחודי" שהוגדר לעיל.

בהמשך מופיע קובץ object לדוגמא בשם ps.ob המתאים לקובץ המקור ps.as

עבור כל תא זיכרון <u>המכיל הוראה (לא נתונים)</u> מופיע בקובץ object מידע עבור תכנית הקישור. מידע זה ניתן ע"י 2 הסיביות הימניות של הקידוד (שדה ה- E,R,A)

האות 'A'(קיצור של absolute) מציינת שתוכן התא אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בזמן ביצועה למשל מילה המכילה אופרנד מיידי).

האות 'R' (קיצור של relocatable) מציינת שתוכן התא כן תלוי במקום בזיכרון שבו ייטען בפועל קוד המכונה של התכנית בזמן ביצועה. לכן יש לעדכן את תוכן התא, בשלב הטעינה, על ידי הוספת היסט (Offset) מתאים (היסט זה הינו המען בו תטען המילה הראשונה של התכנית). במקרה כזה 2 הסיביות הימניות יכילו את הערך 10

האות 'E' (קיצור של external) מציינת שתוכן התא תלוי בערכו של סמל חיצוני (external), וכי רק בזמן הקישור ניתן יהיה להכניס לתא את הערך המתאים. במקרה כזה 2 הסיביות הימניות יכילו את הערך 01.

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 כולם בבסיס 4 הייחודי

M1: .mat [2][2] 1,2,3,4

. (ראו לדוגמה את הקובץ ps.ent המתאים לקובץ המקור ששמו ps.as).

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 4 הייחודי.

<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.

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

```
; file ps.as

.entry LOOP
.entry LENGTH
.extern L3
.extern W
MAIN:       mov     M1[r2][r7],W
            add     r2,STR
LOOP:       jmp     W
            prn     #-5
            sub     r1, r4
            inc     K

            mov     M1[r3][r3],r3
            bne     L3
END:        stop
STR:        .string "abcdef"
LENGTH:     .data   6,-9,15
K:          .data   22
M1:         .mat [2][2]  1,2,3,4
```

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

Label	Decimal Address	Base 4 Address	Instruction	Operands	Binary machine code
MAIN:	**0100**	1210	**mov**	M1[r2][r7],W	**0000-10-01-00**
	0101	1211		כתובת של M1	**10000101-10**
	0102	1212		קידוד אוגרי האינדקסים במטריצה	**0010-0111-00**
	0103	1213		כתובת של W (סמל חיצוני)	**00000000-01**
	0104	1220	**add**	r2, STR	**0010-11-01-00**
	0105	1221		קידוד מספר האוגר	**0010-0000-00**
	0106	1222		כתובת של STR	**01111010-10**
LOOP:	**0107**	1223	**jmp**	W	**1001-00-01-00**
	0108	1230		כתובת של W	**00000000-01**
	0109	1231	**prn**	#-5	**1100-00-00-00**
	0110	1232		המספר 5-	**11111011-00**
	0111	1233	**sub**	r1,r4	**0011-11-11-00**
	0112	1300		קידודי מספרי האוגרים	**0001-0100-00**
	0113	1301	**inc**	K	**0111-00-01-00**
	0114	1302		כתובת של K	**10000100-10**
	0115	1303	**mov**	M1[r3][r3],r3	**0000-10-11-00**
	0116	1310		כתובת של M1	**10000101-10**
	0117	1311		קידוד אוגרי האינדקסים במטריצה	**0011-0011-00**
	0118	1312		קידוד מספר האוגר של היעד	**0000-0011-00**
	0119	1313	**bne**	L3	**1010-00-01-00**
	0120	1320		כתובת של L3 (סמל חיצוני)	**00000000-10**
END:	**0121**	1321	**stop**		**1111-00-00-00**

STR:	**0122**	1322	**.string**	“abcdef”	**0001100001**
	0123	1323			**0001100010**
	0124	1330			**0001100011**
	0125	1331			**0001100100**
	0126	1332			**0001100101**
	0127	1333			**0001100110**
	0128	2000			**0000000000**
LENGTH:	**0129** **0130** **0131**	2001 2002 2003	**.data**	6,-9,15	**0000000110** **1111110111** **0000001111**
K:	**0132**	2010	**.data**	22	**0000010110**
M1	**0133** **0134** **0135** **0136**	2011 2012 2013 2020	**.mat**	[2][2] 1,2,3,4	**0000000001** **0000000010** **0000000011** **0000000100**

הקובץ ps.ob:

כל תוכן הקובץ מיוצג במספרים בבסיס 4 הייחודי.
הערה: שורת הכותרת אינה חלק מהקובץ, ונועדה להבהרה בלבד.

Base 4 address	Base 4 code
bbc	dd
bcba	**aacba**
bcbb	**cabbc**
bcbc	**acbda**
bcbd	**aaaab**
bcca	**acdba**
bccb	**acaaa**
bccc	**bdccc**
bccd	**cbaba**
bcda	**aaaab**
bcdb	**daaaa**
bcdc	**ddcda**
bcdd	**addda**
bdaa	**abbaa**
bdab	**bdaba**
bdac	**cabac**
bdad	**aacda**
bdba	**cabbc**
bdbb	**adada**
bdbc	**aaada**
bdbd	**ccaba**
bdca	aaaab
bdcb	**ddaaa**
bdcc	**abcab**
bdcd	**abcac**
bdda	**abcad**
bddb	**abcba**
bddc	**abcbb**
bddd	**abcbc**
caaa	**aaaaa**
caab	**aaabc**
caac	**dddbd**
caad	**aaadd**
caba	**aabbc**
cabb	**aaaab**
cabc	**aaaac**
cabd	**aaaad**
caca	**aaaba**

קובץ ps.ent:

LOOP	bccd
LENGTH	caab

קובץ ps.ext:

W	bcbd
W	bcda
L3	bdca

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, כגון אישפוז ממושך. במקרים אלו יש לבקש ולקבל אישור **מראש ממנחה הקבוצה**.

ב ה צ ל ח ה !