

האוניברסיטה הפתוחה

20465

## **מעבדה בתכנות מערכות**

חוברת הקורס – סתיו 2023א

כתבה: מיכל אבימור

אוקטובר 2022 – סמסטר סתיו – תשפ"ג

**פנימי – לא להפצה.**

© כל הזכויות שמורות לאוניברסיטה הפתוחה.

## תוכן העניינים

א	אל הסטודנט
ג	1. לוח זמנים ופעילויות
ה	2. תיאור המטלות
ו	3. התנאים לקבלת נקודות זכות
1	ממ"ן 11
5	ממ"ן 12
7	ממ"ן 22
13	ממ"ן 23
15	ממ"ן 14



## 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](http://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](mailto: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 מאוד 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 (מס' קורס 20465 /א2023)**

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	28.10.2022-23.10.2022	ספר C פרקים 1-2-3	מפגש ראשון	
2	04.11.2022-30.10.2022	ספר C פרקים 1-2-3		
3	11.11.2022-6.11.2022	ספר C פרק 4	מפגש שני	
4	18.11.2022-13.11.2022	ספר C פרק 4		
5	25.11.2022-20.11.2022	ספר C פרק 5	מפגש שלישי	ממ"ן 11 20.11.2022
6	02.12.2022-27.11.2022	ספר C פרק 5		
7	09.12.2022-04.12.2022	ספר C פרק 6	מפגש רביעי	
8	16.12.2022-11.12.2022	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	23.12.2022-18.12.2022 (ב-ו חנוכה)	ספר C פרק 6,7	מפגש חמישי	ממ"ן 12 18.12.2022
10	30.12.2022-25.12.2022 (א-ב חנוכה)	ספר C פרק 7		
11	06.01.2023-01.01.2023	ספר C פרק 7	מפגש שישי	
12	13.01.2023-08.01.2023	ספר C פרק 8 + פרויקט		ממ"ן 22 08.01.2023
13	20.01.2023-15.01.2023	פרויקט וחזרה	מפגש שביעי	
14	27.01.2023-22.01.2023	פרויקט וחזרה	מפגש שמיני	ממ"ן 23 22.01.2023  ממ"ן 14** 19.03.2023

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 מנחה הקבוצה.



## 2. תיאור המטלות

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

רוב המטלות בקורס זה הן **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 (.h, .c), 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 יעשה 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.

**לתשומת 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 בשלוש!), 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: 2023'

מועד אחרון להגשה: 20.11.2022

<b>קיימות שתי חלופות להגשת מטלות:</b>
<ul style="list-style-type: none"><li>• שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס</li><li>• שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד</li></ul>
<b>הסבר מפורט ב"נוהל הגשת מטלות מנחה"</b>

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). הקבצים של כל תוכנית יהיו בתיקה נפרדת. נדרש ששם התיקה 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.

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

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

נגדיר מחרוזת מקוצרת באופן הבא:

בהינתן מחרוזת תווים, נחפש בה רצפים עולים של תווים מתוך אותיות האלף-בית האנגלי, שאורכם לפחות 3. לדוגמה, במחרוזת הבאה ישנם שלושה רצפים כאלה, המודגשים בקווים:

dabcefLMNOpQrstuv567zyx

כל רצף יקוצר לשלושה תווים: האות הראשונה ברצף, מקף, והאות האחרונה ברצף. עבור הדוגמה למעלה נקבל את המחרוזת המקוצרת:

da-cefL-OpQr-v567zyx

לתשומת לב: מדובר ברצפים של אותיות האלף-בית בלבד (ולא כל תווים אחרים). אותיות קטנות וגדולות נחשבות שונות זו מזו. הרצף נקבע לפי קודי האסקי המתאימים.

עליכם לכתוב פונקציה המקבלת כפרמטר מחרוזת, והופכת אותה למחרוזת מקוצרת.  
לתשומת לב: הפונקציה אינה בונה מחרוזת חדשה, אלא משנה את המחרוזת המועברת אליה.

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

מותר להניח שהאורך המקסימלי של מחרוזת הקלט הוא 80 תווים.

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.

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

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

## שאלה 2 (תכנית ראשית בקובץ my\_rottate.c) (50 נקודות)

עליכם לכתוב תכנית המכילה את הפונקציה הבאה:

```
unsigned int my_rottate (unsigned int a, int b)
```

פונקציה זו מבצעת רוטציה (סיבוב) של b סיביות על פרמטר a.

הסבר: כאשר מדובר על ביצוע רוטציה של b מקומות על פרמטר a, הכוונה היא להזזת סיביות פרמטר a, b סיביות ימינה, כאשר b חיובי, (ו-b סיביות שמאלה, כאשר b שלילי). כתוצאה מהזזת b סיביות ימינה, "מתרוקנות" b הסיביות השמאליות של פרמטר a. b הסיביות שנזרקו מתוך a מצידו הימני, יוכנסו לתוך a מצידו השמאלי. (בדיוק הפוך כאשר b שלילי. הסיביות ה"נופלות" מהקצה השמאלי של a מוכנסות למקומות הריקים שנוצרו בקצה הימני של a. דוגמה בהמשך).

על הפונקציה להחזיר את הערך לאחר ביצוע הרוטציה. על התוכנית להדפיס את הערך לפני ואחרי השינוי. הערכים יקלטו בבסיס 10, כפי שמוסבר כאן בהמשך. את ההדפסות, לפני ואחרי השינוי, יש לבצע גם בבסיסים 2 8 16, בנוסף לבסיס 10.

הערה: אסור שהמימוש יהיה תלוי מחשב, כלומר לא מובטח כאן שייצוג של משתנה מטיפוס unsigned int יתפוס שני בתים של זיכרון.

לדוגמה : בהינתן משתנה x הבא :

$x = 10101110 \quad 11111011$

אזי  $\text{my\_rotate}(x, 3)$  יתן :

$x = 01110101 \quad 11011111$

ו-  $\text{my\_rotate}(x, -3)$  יתן :

$x = 01110111 \quad 11011101$

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.

על התכנית להדפיס הודעת בקשה ידידותית לקלט. כמו כן יש להדפיס באופן נאה את הנתונים כפי שנקלטו, 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. במידה ותשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

לתשומת לבכם : לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או 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 ובאופן חלקי 6

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

סמסטר: 2023 מועד אחרון להגשה: 18.12.2022

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- 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), קובץ ההרצה (את קבצי 0. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קובץ makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר). קבצי התוכנית יהיו בתיקה. נדרש ששם התיקה 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 האוי"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן תקין.

שאלה 1 (תכנית ראשית בקובץ my\_bcmp.c)

עליכם לממש את הפונקציה הבאה:

```
int my_bcmp (const void *b1, const void *b2, int len);
```

b1 – מצביע לקטע זיכרון ראשון.

b2 – מצביע לקטע זיכרון שני.

len – מספר הבתים (bytes) להשוואה.

מטרת הפונקציה: להשוות את len הבתים, החל מהמקום אליו מצביע b1, אל len הבתים, החל מהמקום אליו מצביע b2.

הערך המוחזר: 0 אם נמצא ששני קטעי הזיכרון מכילים תוכן זהה, ואחרת ערך כלשהו שונה מ-0.

הערות:

- קטעי הזכרון יכולים להיות חופפים זה לזה באופן חלקי או מלא

- קטעי זיכרון באורך 0 הם תמיד זהים זה לזה
- לא ניתן להניח שקטעי הזיכרון מסתיימת בתו '0', ואסור לפונקציה לשנות את הקטעים.

לצורך בדיקת הפונקציה, נשתמש בקטעי זיכרון המוזנים לתכנית על ידי המשתמש, באופן הבא: התכנית תקבל כקלט מחרוזת, שני אינדקסים לתוך המחרוזת, ואת הערך len. האינדקסים מייצגים מצביעים לקטעי הזיכרון (כמובן שיש לבנות מהאינדקסים מצביעים לצורך הקריאה לפונקציה).

סדר הנתונים בקלט הוא: len, אחריו שני האינדקסים, ולבסוף המחרוזת. מותר להניח שהאורך המקסימלי של המחרוזת הוא 512. יש לקלוט את כל אחד מארבעת הנתונים בנפרד. לתשומת לב: len אינו אורך מחרוזת הקלט, אלא מספר הבתים להשוואה.

יש לכתוב תכנית ראשית (הפונקציה main) שתבצע קלט מהמשתמש, תפעיל את הפונקציה my\_bcmp, ותדפיס באופן נאה את התוצאה המוחזרת מן הפונקציה.

חובה לבצע בדיקות תקינות של הקלט טרם הקריאה לפונקציה. במקרה של שגיאה כלשהי בנתוני הקלט, על התכנית להפסיק את עבודתה, ולהדפיס הודעת שגיאה מפורטת, לרבות מידע על הנתון השגוי ככל הניתן. להלן דוגמאות של שגיאות אפשריות (אפשר להוסיף עוד שגיאות):

- אחד האינדקסים ו/או הערך len אינו מספר עשרוני שלם אי-שלילי
- אחד מקטעי הזיכרון גולש מגבולות המחרוזת (לפי ערכי האינדקס ו- len)

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.

על התכנית להדפיס הודעת בקשה ייחודית לקלט, המפרטת מה על המשתמש להקליד. כמו כן יש להדפיס באופן נאה את הנתונים שנקלטו, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

אין להגביל את הקלט באופן כלשהו. כלומר, המשתמש יכול להעביר כרצונו כל כמות ערכים בשורת הקלט.

חובה לצרף להגשה מספר הרצות בדיקה המדגימות את פעולת התכנית על גדלים שונים של נתוני הקלט, לרבות הדגמה של הטיפול במגוון השגיאות בקלט. יש להגיש תדפיסי מסך (או קבצי פלט) של כל הרצות הדוגמה. במידה ותשתמשו בקבצי קלט, יש להגיש גם קבצים אלה

לתשומת לבכם: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או 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: 2023 א' מועד אחרון להגשה: 08.01.2023

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני, באישור המנחה בלבד
- 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 הנחיות במטלה/במפגש/באתר). קבצי התוכנית יהיו בתיקה. נדרש ששם התיקה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה `main`, ללא הסיומת `.c`. יש להגיש תכנית מלאה (בין השאר מכילה `main`), הניתנת להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות 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 הוגשו באופן תקין.

## שאלה 1 (תכנית ראשית בקובץ `myset.c`, ובנוסף הקבצים `set.c`, `set.h`)

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

תזכורת: קבוצה (`set`) היא אוסף של איברים, בני מניה, וללא חזרות (כלומר, לכל איבר בקבוצה יש ערך שונה מכל האיברים האחרים).

### משימות התכנית:

עליכם לכתוב תכנית מחשב הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן. הפקודות עוסקות בפעולות על קבוצות.

עליכם להגדיר, תוך שימוש ב- `typedef`, את הטיפוס `set`, אשר מחזיק קבוצה של מספרים שלמים מהתחום הסגור `[0...127]`. על הטיפוס (מבנה הנתונים) `set` להיות חסכוני מבחינת כמות הזיכרון הנדרשת. כך למשל, מערך של 128 בתים אינו חסכוני. רמז: אפשר להסתפק בסיבית (`bit`) אחת לכל איבר בקבוצה.

בנוסף עליכם להגדיר, בתכנית הראשית, שישה משתנים מטיפוס set, בשמות הבאים :  
SETA, SETB, SETC, SETD, SETE, SETF.  
בתחילת ריצת התכנית, יש לאתחל כל אחד מהמשתנים לקבוצה הריקה.

כעת, עליכם לבצע פעולות מגוונות על קבוצות. כל פעולה תופעל באמצעות פקודה שמועברת מהמשתמש בקלט לתכנית.

בפקודות שיפורטו להלן, אופרנד שהוא שם של קבוצה, יהיה אחד מששת המשתנים שהוגדרו לעיל.

מבנה הפקודות המשמשות כקלט לתכנית :

**לתשומת לב :** סדר קריאת השדות בפקודה הוא משמאל לימין.

### 1. הצבת איברים בקבוצה

**רשימת-ערכים-מופרדים-זה-מזה-בפסיקים, שם-קבוצה read\_set**

הפקודה מכניסה את הערכים שברשימה לתוך הקבוצה ששמה ניתן בפקודה. רשימת הערכים אינה סדורה, ומותר לערך להופיע בה יותר מפעם אחת (מופעים חוזרים לא יילקחו בחשבון). סוף רשימת הערכים מסומן על ידי המספר השלילי -1.

לדוגמה, הפקודה : read\_set SETA, 5, 6, 5, 76, 44, 23, 23, 98, 23, -1

תציב במשתנה SETA את הקבוצה : 5, 6, 23, 44, 76, 98

לתשומת לב : ההצבה יוצרת קבוצה חדשה, שמחליפה את התוכן הקודם של המשתנה. אם הרשימה אינה מכילה אף ערך (מלבד -1 המסיים), הקבוצה שתיווצר היא ריקה.

### 2. הדפסת קבוצה

**שם-קבוצה print\_set**

הפקודה מדפיסה את איברי הקבוצה ששמה ניתן בפקודה, בסדר עולה של הערכים, ולכל היותר 16 ערכים בכל שורת פלט. יש להקפיד על פורמט נאה של ההדפסה.  
אם הקבוצה ריקה, יש להדפיס "The set is empty".

### 3. איחוד של קבוצות

**שם-קבוצה-ג', שם-קבוצה-ב', שם-קבוצה-א' union\_set**

הפקודה מבצעת איחוד של קבוצה א' עם קבוצה ב', ואת התוצאה מאחסנת בקבוצה ג'.  
תזכורת : תוצאת האיחוד היא קבוצת כל האיברים הנמצאים בקבוצה א' ו/או בקבוצה ב'.

### 4. חיתוך של קבוצות

**שם-קבוצה-ג', שם-קבוצה-ב', שם-קבוצה-א' intersect\_set**

הפקודה מבצעת חיתוך של קבוצה א' עם קבוצה ב' ואת התוצאה מאחסנת בקבוצה ג'.  
תזכורת : תוצאת החיתוך היא קבוצת כל האיברים הנמצאים גם בקבוצה א' וגם בקבוצה ב'.

### 5. חיסור של קבוצות

**שם-קבוצה-ג', שם-קבוצה-ב', שם-קבוצה-א' sub\_set**

הפקודה מבצעת חיסור של קבוצה ב' מקבוצה א', ואת התוצאה מאחסנת בקבוצה ג'.  
תזכורת : תוצאת החיסור היא קבוצת כל האיברים הנמצאים בקבוצה א' ולא נמצאים בקבוצה ב'.

## 6. הפרש סימטרי של קבוצות

שם-קבוצה-ג', שם-קבוצה-ב', שם-קבוצה-א' symdiff\_set

הפקודה מחשבת הפרש סימטרי של קבוצה א' וקבוצה ב', ואת התוצאה מאחסנת בקבוצה ג'.  
תזכורת: הפרש סימטרי הוא קבוצת כל האיברים הנמצאים בקבוצה א' או בקבוצה ב', אבל לא נמצאים בחיתוך של קבוצה א' עם קבוצה ב'.

## 7. עצירת התכנית

stop

זוהי פקודה ללא אופרנדים, הגורמת לסיום מידי של התכנית.

לתשומת לב: אותו שם קבוצה יכול לשמש ביותר מאופרנד אחד באותה הפקודה. מימוש הפעולות על קבוצות צריך להתחשב באפשרות זו (לא לדרוס נתונים תוך כדי חישוב).  
לדוגמה, הפעולות שלהלן תקינות ומוגדרות היטב:

union\_set SETC, SETD, SETD

intersect\_set SETA, SETF, SETA

sub\_set SETC, SETC, SETC (מה האפקט של פעולה זו?)

union\_set SETA, SETA, SETF (מה האפקט של פעולה זו?)

המבנה התחבירי של הקלט:

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

אופן פעולת התכנית:

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

התכנית אינה 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. אין לעצור את התכנית עם גילוי השגיאה הראשונה. אין צורך לדווח על יותר משגיאה אחת בכל שורת קלט.

יש לטפל גם במצב של EOF (גמר הקלט). עצירת התכנית שלא באמצעות פקודת stop מפורשת בקלט 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'.

להלן דוגמאות של קלט שגוי:

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

1. לפקודה :  
`read_set SETG, 3, 6, 5, 4, 4, -1`  
 Undefined set name  
 יש להגיב בהודעה כגון :
2. לפקודה :  
`read_set setA, 3, 6, 5, 4, 4, -1`  
 Undefined set name  
 יש להגיב בהודעה כגון :
3. לפקודה :  
`do_it SETA, SETB, SETC`  
 Undefined command name  
 יש להגיב בהודעה כגון :
4. לפקודה :  
`UNION_set SETA, SETB, SETC`  
 Undefined command name  
 יש להגיב בהודעה כגון :
5. לפקודה :  
`read_set SETB, 45, 567, 34, -1`  
 Invalid set member – value out of range  
 יש להגיב בהודעה כגון :
6. לפקודה :  
`read_set SETA, 45, 56, 45, 34`  
 List of set members is not terminated correctly  
 יש להגיב בהודעה כגון :
7. לפקודה :  
`read_set SETA, 45, -3, 2, 45, 34, -1`  
 Invalid set member – value out of range  
 יש להגיב בהודעה כגון :
8. לפקודה :  
`read_set SETA, 45, 2, xyz, 34, -1`  
 Invalid set member – not an integer  
 יש להגיב בהודעה כגון :
9. לפקודה :  
`read_set SETA, 45, 2, 24.0, 34, -1`  
 Invalid set member – not an integer  
 יש להגיב בהודעה כגון :
10. לפקודה :  
`union_set SETC, SETA`  
 Missing parameter  
 יש להגיב בהודעה כגון :
11. לפקודה :  
`union_set SETC, SETA, SETB,`  
 Extraneous text after end of command  
 יש להגיב בהודעה כגון :
12. לפקודה :  
`print_set SETC, SETD`  
 Extraneous text after end of command  
 יש להגיב בהודעה כגון :

13. לפקודה :

sub\_set SETF, , SETD, SETA

יש להגיב בהודעה כגון :

Multiple consecutive commas

14. לפקודה :

intersect\_set SETF SETD SETA

יש להגיב בהודעה כגון :

Missing comma

15. לפקודה :

symdiff\_set, SETF, SETB, SETA

יש להגיב בהודעה כגון :

Illegal comma

להלן דוגמה של סדרת פקודות שכולן תקינות :

הערה : סידרה כגון זו יכולה לשמש כקלט להרצת בדיקה של נכונות הביצוע של הפקודות (ללא טיפול בשגיאות בקלט).

```
print_set SETA
print_set SETB
print_set SETC
print_set SETD
print_set SETE
print_set SETF
read_set SETA, 45, 23, 6, 7, 4, 3, 75, 45, 34, -1
print_set SETA
read_set SETB, 5, 4, 3, 2, 78, 45, 43, -1
print_set SETB
read_set SETC, 100, 105, 101, 103, 104, -1
print_set SETC
read_set SETC, 127, 0, 3, 126, 127, 0, -1
print_set SETC
read_set SETC, -1
print_set SETC
read_set SETD, -1
print_set SETD
read_set SETC, 110, 111, 112, -1
print_set SETC
union_set SETA, SETB, SETD
print_set SETD
intersect_set SETA, SETB, SETE
print_set SETE
sub_set SETA, SETB, SETF
print_set SETF
symdiff_set SETA, SETB, SETF
print_set SETF
intersect_set SETA, SETC, SETD
print_set SETD
```

```

union_set SETB, SETB, SETE
print_set SETE
intersect_set SETB, SETA, SETB
print_set SETB
union_set SETA, SETC, SETC
print_set SETC
symdiff_set SETC, SETA, SETC
print_set SETC
sub_set SETC,SETC,SETC
print_set SETC
union_set SETF , SETC , SETF
print_set SETF
stop

```

### ארגון קוד התכנית :

יש לחלק את התוכנית למספר קבצי מקור : set.c , myset.c , ו- set.h.

- בקובץ set.c יש לרכז את הפעולות על קבוצות. לכל פעולה יש לממש פונקציה נפרדת, ששמה כשם הפקודה בקלט (לדוגמה : union\_set , print\_set , read\_set , וכד'). האופרנדים של הפעולה יועברו כפרמטרים של הפונקציה, לפי מפרט הפעולה כמוגדר לעיל. לא יבוצע כל קלט/פלט בקובץ זה, ולא ניתוח תחבירי של הקלט.
  - בקובץ myset.c תהיה הפונקציה main, וכן כל פעילויות האינטראקציה עם המשתמש, ניתוח הפקודות, והדפסת הודעות השגיאה. כמו כן, בקובץ זה יוגדרו ששת המשתנים מטיפוס set.
  - בקובץ set.h תהיה הגדרת טיפוס הנתונים set, וכן ההצהרות (אב טיפוס) של הפונקציות הממומשות בקובץ set.c. יש לכלול (#include) את הקובץ set.h בקבצי המקור האחרים.
  - אפשר 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 כדיבוג התכנית. בכל קובץ קלט תהיה סדרה של פקודות על קבוצות.
- על התכנית להדפיס הודעת בקשה ידידותית לקלט עבור כל שורת קלט (כל פקודה). כמו כן, לפני הניתוח של שורת הקלט, על התכנית להדפיס 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 הפעולות על קבוצות ובכל ששת הקבוצות המוגדרות, וכן את הטיפול בכל מגוון השגיאות בקלט. רמז : מומלץ להכניס בקלט פקודת הדפסה של קבוצת התוצאה אחרי כל פעולה, כדי להראות שהתוצאה אכן נכונה (ראו לעיל הדוגמה של סדרת פקודות תקינות).
- יש להגיש תדפיס מסך (או קובץ פלט) של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

לתשומת לבכם : לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או 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 6,7,8

משקל המטלה: 12 נקודות (רשות)

מספר השאלות: 2

מועד אחרון להגשה: 22.01.2023

סמסטר: 2023א'

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- 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), קובץ ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קובץ `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`. 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.

5 נק') א. קובץ הקלט הסטנדרטי `stdin` הוא תמיד המקלדת, קובץ הפלט הסטנדרטי `stdout` הוא תמיד המסך, וקובץ השגיאות הסטנדרטי `stderr` הוא תמיד המסך.

5 נק') ב. אם נגדיר משתנה כ-`static register`, נוכל לייעל את זמן השימוש במשתנה, וכן לשמור על ערכו מקריאה לקריאה.

את הפתרון לשאלה זו יש להגיש במסמך (קובץ) מוקלד, בכל פורמט.

## שאלה 2 (90 נקודות) (תכנית ראשית בקובץ permut.c)

הגדרה: **תמורה (permutation)** של **מחרוזת תווים** היא מחרוזת המתקבלת על ידי ערבוב כלשהו של סדר התווים במחרוזת המקורית.

עליכם לכתוב תכנית המקבלת שני ארגומנטים בשורת הפקודה: שם של קובץ קלט, ומחרוזת תווים. על התוכנית לסרוק את תוכן הקובץ, ולהדפיס לפלט הסטנדרטי את כל התמורות של המחרוזת אשר תימצאנה בקובץ.

התמורות תודפסנה לפי סדר הופעתן בקובץ, כל תמורה תודפס בשורה חדשה. ייתכנו מופעים מרובים של כל תמורה, ויש להדפיס את כולם. הניח שאין בקובץ תמורות שחופפות זו לזו.

לדוגמה: אם שורת הפקודה נראית כך:

```
> prog data.in chair
```

ותוכן הקובץ data.in הוא:

```
I am a chair
You are an arich and not a CHAIR
We are haircs and not gariches
Goodbye
```

הפלט של התוכנית יהיה:

```
chair
arich
hairc
arich
```

על התוכנית להדפיס הודעת שגיאה מתאימה לקובץ **השגיאות** הסטנדרטי, ולהפסיק את עבודתה, במקרים הבאים: מספר הארגומנטים בשורת הפקודה אינו כנדרש; בעיה בפתיחת הקובץ; הקובץ ריק.

במקרה והתוכנית סרקה את כל תוכן הקובץ אך לא מצאה אף תמורה, עליה להדפיס הודעה מתאימה לקובץ **הפלט** הסטנדרטי, ולסיים את ריצתה.

**חובה לצרף להגשה מספר הרצות בדיקה** המדגימות את פעולת התכנית, לרבות הטיפול בשגיאות. יש להגיש **קובץ קלט + תדפיס מסך** של כל הרצת דוגמה.

**לתשומת לבכם**: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או 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 : 2023 מועד אחרון להגשה : 19.03.2023

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- 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 וכתובה נאה ומובנית.

נזכיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של מבני הנתונים : רצוי (ככל האפשר) להפריד בין הגישת 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 ותייעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

מותר להשתמש בפרויקט בכל מגוון הספריות הסטנדרטיות של שפת C, אבל אין להשתמש בספריות חיצוניות אחרות.

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 אותן 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 - אוגרים) 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 וחסור 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. כמוכן 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 יעודית משלו, ובהתאם גם שפת אסמבלי יעודית משלו. לפיכך, גם האסמבלר (כלי התרגום) הוא יעודי 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: תאור מודל המחשב להלן הוא חלקי בלבד, ככל שנחוצ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזית), אוגרים (רגיסטרים) 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 14 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 13. 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 14 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממוספרות כמו ברגיסטר.

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

### מבנה הוראת המכונה:

**כל הוראת מכונה מקודדת למספר מילות זיכרון**, החל ממילה אחת ועד למקסימום ארבע מילים, בהתאם לשיטות המיעון בהן נעשה שימוש (ראו בהמשך). בכל סוגי ההוראות, המבנה של המילה הראשונה זהה. מבנה המילה הראשונה בהוראה הוא כדלהלן:

13 12	11 10	9 8 7 6	5 4	3 2	1 0
פרמטר1	פרמטר2	opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד כל מילה בקוד המכונה ייעשה בבסיס 2 **ייחודי** המוגדר כדלקמן: 0 שקול לתו נקודה (=0), ו-1 שקול לתו / (≠1). ראו דוגמאות בסוף הגדרת הפרויקט.

**סיביות 6-9** במילה הראשונה של ההוראה מהוות את קוד הפעולה (opcode). כל opcode מיוצג בשפת אסמבלי על ידי "שם פעולה". בשפה שלנו יש 16 קודי פעולה והם:

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

שמות הפעולות נכתבים תמיד באותיות קטנות. פרוט המשמעות של הפעולות יבוא בהמשך.

### סיביות 0-1 (A,R,E)

סיביות אלה מציינות 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.

**סיביות 10-13** רלוונטיות רק אם מדובר בשיטת מיעון מספר 2 (ראו הסבר בהמשך), ואחרת ערכן הוא 0. הסיביות מאפיינות שני פרמטרים. סיביות 10-11 אחראיות על פרמטר 2, ואילו סיביות 12-13 אחראיות על פרמטר 1. הסבר מפורט מופיע בהמשך, בדיון על שיטות מיעון.  
אם הפרמטר הוא מספר מידי, הסיביות יקבלו 00.  
אם הפרמטר הוא רגיסטר, הסיביות יקבלו 11.  
אם הפרמטר הוא שם של תווית, הסיביות יקבלו 01.

להלן תיאור של שיטות המיעון במכונה שלנו :

מס'	שיטת מיעון	תוכן המילים הנוספות	אופן הכתיבה	דוגמה
0	מיעון מידי	המילה הנוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 12 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 מידי. ההוראה כותבת את הערך 1- לתוך רגיסטר r2
1	מיעון ישיר	המילה הנוספת של ההוראה מכילה מען של מילה בזיכרון. מילה זו בזיכרון הינה האופרנד. המען מיוצג ב- 12 סיביות אליהן מתווספות זוג סיביות של שדה A,R,E.	האופרנד הינו <u>תווית</u> שכבר הוצהרה או שתוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בתחילת הנחיית 'data' או 'string', או בתחילת ההוראה של התוכנית, או באמצעות אופרנד של הנחיית 'extern'.	נתונה למשל ההגדרה : x: .data 23  אפשר לכתוב הוראה : dec x  בדוגמה זו, ההוראה מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x).
2	מיעון קפיצה עם פרמטרים	שיטת מיעון זו רלוונטית וניתנת לשימוש רק בפעולות קפיצה (jmp,bne,jsr).  השיטה מיועדת להעברה נוחה של שני פרמטרים לצורך שימוש ביעד הקפיצה. הפרמטר הראשון יועבר ברגיסטר r6, והשני ברגיסטר r7 (התוכן הקודם של רגיסטרים אלה נדרס).  בשיטת מיעון זו יש לכל היותר 3 מילות מידע נוספות בקוד ההוראה. המילה הראשונה הנוספת היא כתובת התווית אליה קופצים. המילה הנוספת השניה היא קידוד הפרמטר הראשון. המילה הנוספת השלישית היא קידוד הפרמטר השני. ייתכן ושני הפרמטרים יקודדו למילה משותפת, ואז יהיו רק 2 מילות מידע נוספות.  לכל אחת מהמילים הנוספות של שיטת המיעון מתווספות זוג סיביות של שדה A,R,E.	האופרנד מורכב משם של תווית אליה רוצים לקפוץ, ואחריה בתוך סוגריים שני הפרמטרים, מופרדים זה מזה בפסיק. אין רווחים בין מרכיבי האופרנד. לא ניתן לציין יותר או פחות משני פרמטרים.  כל פרמטר יכול להיות מספר מידי (בדומה לשיטת מיעון 0), או תווית (בדומה לשיטת מיעון 1), או רגיסטר (בדומה לשיטת מיעון 3 – ראו הגדרה בהמשך הטבלה).  הסיביות של כל פרמטר במילת הקוד הראשונה ייקבעו בהתאם לסוג הפרמטר. המילה הנוספת תקודד בדומה לשיטת המיעון המקבילה.  אם הפרמטר הראשון הוא רגיסטר, הוא יקודד במילה נוספת, כמו רגיסטר מקור בשיטת מיעון 3. אם הפרמטר השני רגיסטר, הוא יקודד כמו רגיסטר יעד.	jmp L1(#5,N) בדוגמה זו, הקבוע 5 מקודד במילה השלישית של ההוראה, והכתובת המיוצגת על ידי התווית N מקודדת במילה הרביעית.  ההוראה מבצעת קפיצה לתווית L1, ומועברים שני פרמטרים : הקבוע 5 (ברגיסטר r6), והמילה שבכתובת N (ברגיסטר r7).  או למשל, jsr L1(r3,r5)  בדוגמה זו, שני הפרמטרים הם רגיסטרים, והם חולקים את המילה השלישית של קוד ההוראה.

		אם שני הפרמטרים הם רגיסטרים, הם יחלקו מילה משותפת.		
3	מיעון רגיסטר ישיר	האופרנד הוא רגיסטר. אם הרגיסטר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בששת הסיביות 2-7 את מספרו של הרגיסטר. אם הרגיסטר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 8-13 את מספרו של הרגיסטר. אם בפקודה יש שני אופרנדים, ושניהם רגיסטרים, הם יחלקו מילה נוספת אחת משותפת, כאשר הסיביות 2-7 הן עבור אוגר היעד, והסיביות 8-13 עבור רגיסטר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0.	האופרנד הינו שם של רגיסטר.	mov r1,r2  בדוגמה זו, ההוראה מעתיקה את תוכן רגיסטר r1 לתוך אוגר r2.  בדוגמה זו שני האופרנדים הם רגיסטרים, אשר יקודדו למילה נוספת משותפת.

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

#### מפרט הוראות המכונה:

בתיאור הוראות המכונה נשתמש במונח PC (קיצור של "Program Counter"). זהו אוגר פנימי של המעבד (לֹא 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 הנדרשים לפעולה.

#### **קבוצת ההוראות הראשונה:**

אלו הן הוראות המקבלות שני אופרנדים.

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

הוראה	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק תוכן המשתנה A (המילה שבכתובת A בזכרון) לרגיסטר r1.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן 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 דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	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 רגיסטר r0.
sub	sub #3, r1	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני)	רגיסטר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	lea HELLO, r1	lea הינו ראשי תיבות של load effective address. פעולה זו מבצעת טעינה של המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	המען שמייצגת התווית HELLO מוכנס לרגיסטר r1.

### קבוצת ההוראות השניה:

הוראות הדורשות אופרנד אחד בלבד. במקרה זה זוג הסיביות 4-5 במילה הראשונה של קידוד ההוראה הן חסרות משמעות, מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). לפיכך הסיביות 4-5 יכילו תמיד 0.  
ההוראות השייכות לקבוצה זו הן:  
not, clr, inc, dec, jmp, bne, red, prn, jsr

הוראה	הפעולה המתבצעת	דוגמה	הסבר דוגמה
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	$r2 \leftarrow \text{not } r2$
clr	איפוס תוכן האופרנד.	clr r2	$r2 \leftarrow 0$
inc	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
dec	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$
jmp	קפיצה בלתי מותנית (עם או בלי פרמטרים) אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.  אם ההוראה כוללת פרמטרים, ערכי הפרמטרים מועתקים לרגיסטרים r6, r7.	jmp LINE  או  jmp LINE(#-6,r4)	$PC \leftarrow \text{LINE}$  $r6 \leftarrow -6$ $r7 \leftarrow r4$
bne	bne הינו ראשי תיבות של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית (עם או בלי פרמטרים). מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z ברגיסטר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בהוראת cmp.  אם ההסתעפות מתבצעת, וההוראה כוללת פרמטרים, ערכי הפרמטרים יועתקו לרגיסטרים	bne LINE  או  bne LINE(X,r4)	אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הינו 0: $PC \leftarrow \text{LINE}$  אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הינו 0: $PC \leftarrow \text{LINE}$ $r6 \leftarrow \text{word at X}$ $r7 \leftarrow r4$

		r6, r7. אחרת, הרגיסטרים נשארים ללא שינוי.	
קוד ה-ascii של התו הנקרא מהקלט הסטנדרטי יוכנס לרגיסטר r1.	red r1	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red
התו אשר קוד ה-ascii שלו נמצא ברגיסטר r1 יודפס לפלט הסטנדרטי.	prn r1	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn
push(PC) PC ← FUNC  push(PC) PC ← FUNC r6 ← 75 r7 ← word at X	jsr FUNC  או  jsr FUNC(#75,X)	קריאה לשגרה (סברוטניה), עם או בלי פרמטרים. מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזכרון המחשב, והאופרנד מוכנס ל-PC.  אם ההוראה כוללת פרמטרים, ערכי הפרמטרים מועתקים לרגיסטרים r6, r7.	jsr

#### קבוצת ההוראות השלישית:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 2-5) במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: stop, rts.

הוראה	הפעולה המתבצעת	דוגמה	הסבר דוגמה
rts	מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC).	rts	ההוראה הבאה שתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה.
stop	עצירת ריצת התוכנית.	stop	התכנית עוצרת מיידית

#### מבנה תכנית בשפת אסמבלי:

תכנית בשפת אסמבלי בנויה **ממקראים וממשפטים** (statements).

#### מקראים:

מקראים הם קטעי קוד הכוללים בתוכם משפטים. בתוכנית ניתן להגדיר מקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במקרו ממקום מסוים בתוכנית יגרום לפרישת המקרו לאותו מקום.

הגדרת מקרו נעשית באופן הבא: (בדוגמה שם המקרו הוא m1)

```
mcr m1
inc r2
mov A,r1
endmcr
```

שימוש במקרו הוא פשוט אזכור שמו.

למשל, אם בתוכנית במקום כלשהו כתוב :

```
.  
.   
.   
m1  
.   
.   
m1  
.   
.   
.
```

בדוגמה זו, השתמשנו פעמיים במקרו m1, התוכנית לאחר פרישת המקרו תיראה כך :

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

### התוכנית לאחר פרישת המקרו היא התוכנית שהאסמבלר אמור לתרגם.

#### הנחיות לגבי מאקרו :

- אין במערכת הגדרות מאקרו מקוננות.
- שם של הוראה או הנחיה לא יכול להיות שם של מאקרו.
- ניתן להניח שלכל שורת מאקרו בקוד המקור קיימת סגירה עם שורת endmcr (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקריאה למאקרו
- 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.

#### משפטים :

קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע בשורה נפרדת. 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 מכילה רק את התווים 't' ו-' ' (טאבים ורווחים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הוא ';' (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 זכרון ואתחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב משם ההוראה (פעולה) שעל המעבד לבצע, והאופרנדים של ההוראה.

כעת נפרט לגבי סוגי המשפטים השונים.

### משפט הנחיה:

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: למילות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 14 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 כפי 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 אחד 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.

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

#### 4. ההנחיה 'extern'.

להנחיה 'extern' פרמטר אחד, 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).

**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

תוויות:

תווית היא 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 הנחייה, או שם של רגיסטר) אינן יכולות לשמש גם כשם של תווית.

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. תווית 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 של ספרות בבסיס עשרוני. לדוגמה: 123, -5, 76 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. זהו 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.

שתי הסיביות בשדה A,R,E יכילו את אחד הערכים הבינאריים: 00, 10, או 01. 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 מיידית). במקרה זה שתי הסיביות הימניות יכילו את הערך 00.

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). במקרה זה שתי הסיביות הימניות יכילו את הערך 10.

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). במקרה זה שתי הסיביות הימניות יכילו את הערך 01.

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

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

```

MAIN:      mov    r3,LENGTH
LOOP:      jmp     L1(#-1,r6)
           mcr     m1
           sub     r1,r4
           bne     END
           endmcr
           prn     #-5
           bne     LOOP(r4,r3)
           m1
L1:         inc     K
           bne     LOOP(K,STR)
END:        stop
STR:        .string "abcdef"
LENGTH:    .data   6,-9,15
K:          .data   22

```

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

```

MAIN:      mov    r3,LENGTH
LOOP:      jmp     L1(#-1,r6)
           prn     #-5
           bne     LOOP(r4,r3)
           sub     r1,r4
           bne     END
L1:         inc     K
           bne     LOOP(K,STR)
END:        stop
STR:        .string "abcdef"

```



LENGTH: .data 6,-9,15  
K: .data 22

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 m1)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא " mcr " (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "יש mcr".
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה m1).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).  
אם דגל "יש mcr" דולק ולא זוהתה תווית endmcr הכנס את השורה לטבלת המאקרו ומחק את השורה הנ"ל מהקובץ. אחרת (לא מאקרו) חזור ל- 1.
7. האם זוהתה תווית endmcr? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל "יש mcr". חזור ל- 1. (סיום שמירת הגדרת מאקרו).
9. סיום: שמירת קובץ מקרו 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 האוגרים, בונים את קוד המכונה.

עליו להחליף את שמות הפעולות mov, add, jmp, prn, sub, inc, bne, stop בקוד הבינארי השקול להם במודל המחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים K, STR, LENGTH, MAIN, LOOP, END במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטען בזיכרון החל ממען 0100 (בבסיס 10). במקרה זה נקבל את ה"תרגום" הבא:

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: mov r3 ,LENGTH	first word of instruction source register r3 address of label LENGTH	00000000110100 00001100000000 00001000001010
0103 0104 0105 0106	LOOP: jmp L1(#-1,r6)	address of label L1 immediate #-1 (2's complement) immediate r6	00111001001000 00000111010010 11111111111100 00000000011000
0107 0108	prn #-5	immediate #-5	00001100000000 11111111101100
0109 0110 0111	bne LOOP(r4,r5)	address of LOOP registers r4,r5	11111010001000 00000110011110 00010000010100
0112 0113	sub r1, r4	registers r1,r4	00000011111100 00000100010000
0114 0115	bne END	address of label END	00001010000100 00000111101010

0116	L1: inc K		00000111000100
0117		address of label K	00001000010110
0118	bne LOOP(K,STR)		01011010001000
0119		address of label LOOP	00000110011110
0120		address of label K	00001000010110
0121		address of label STR	00000111101110
0122	END: stop		00001111000000
0123	STR: .string "abcdef"	ascii code 'a'	00000001100001
0124		ascii code 'b'	00000001100010
0125		ascii code 'c'	00000001100011
0126		ascii code 'd'	00000001100100
0127		ascii code 'e'	00000001100101
0128		ascii code 'f'	00000001100110
0129		ascii code '\0' (end of string)	00000000000000
0130	LENGTH: .data 6,-9,15	integer 6	00000000000110
0131		integer -9 (2's complement)	11111111101111
0132		integer 15	00000000001111
0133	K: .data 22	integer 22	00000000010110

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 הקידוד 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 0122 (עשרוני), אלא רק לאחר שנקראו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים, עם הערכים המספריים המשוויכים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של ההוראות 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 דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	116
END	122
STR	123
LENGTH	130
K	133

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

לתשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התוכנית 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 ישויד 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 0, ולכן נטענת ההוראה הראשונה במען 0. 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 קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסמבלר כל שם פעולה בקידוד שלה. כמו כן, 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 המשויד 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 היא 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, אם כי בקובץ הקלט אין חובה שתהיה הפרדה כזו.** בהמשך מתואר אלגוריתם של האסמבלר, ובו פרטים כיצד לבצע את ההפרדה.

## גילוי שגיאות בתכנית המקור

כפי שהוסבר למעלה, הנחת המטלה היא שאין שגיאות בהגדרות המקור, ולכן שלב קדם האסמבלר אינו מכיל שלב גילוי שגיאות, 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 אוגר 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 מקרו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקארו. 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:

שם ההוראה	שיטות מיעון חוקיות עבור אופרנד המקור	שיטות מיעון חוקיות עבור אופרנד היעד
mov	0,1,2,3	1,2,3
cmp	0,1,2,3	0,1,2,3
add	0,1,2,3	1,2,3
sub	0,1,2,3	1,2,3
not	אין אופרנד מקור	1,2,3
clr	אין אופרנד מקור	1,2,3
lea	1,2	1,2,3
inc	אין אופרנד מקור	1,2,3

1,2,3	אין אופרנד מקור	dec
1,2,3	אין אופרנד מקור	jmp
1,2,3	אין אופרנד מקור	bne
1,2,3	אין אופרנד מקור	red
0,1,2,3	אין אופרנד מקור	prn
1,2,3	אין אופרנד מקור	jsr
אין אופרנד יעד	אין אופרנד מקור	rts
אין אופרנד יעד	אין אופרנד מקור	stop

### 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 חלקים: תמונת ההוראות (code) ותמונת הנתונים (data). לכל חלק יש מונה משלו, ונסמנם IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter).

כמו כן, נסמן ב-L את מספר המילים שתופס קוד המכונה של הוראה נתונה.

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

### מעבר ראשון

1. אתחל  $DC \leftarrow 0$ ,  $IC \leftarrow 0$ .
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם השדה הראשון 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 סימון (סמל מסוג data)-ערכו יהיה DC (אם הסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, עדכן את מונה הנתונים DC בהתאם לאורכם, חזור ל-2.
8. האם זו הנחיית extern או הנחיית entry? אם לא, עבור ל-11.
9. האם זוהי הנחיית extern? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם סימון (סמל מסוג external).
10. חזור ל-2.
11. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג code). ערכו יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את הקוד הבינארי של המילה הראשונה של הפקודה.
14. עדכן  $IC \leftarrow L + IC$ .
15. חזור ל-2.
16. אם נמצאו שגיאות בקובץ המקור, עצור.
17. עדכן בטבלת הסמלים את ערכם של הסמלים מסוג data, ע"י הוספת הערך הסופי של IC (ראו הסבר בהמשך).
18. התחל מעבר שני.

## מעבר שני

1. אתחל  $IC \leftarrow 0$
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-10.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הנחייה .extern, .string, .data? אם כן, חזור ל-2.
5. האם זוהי הנחייה .entry? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים כ-*entry*. חזור ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השניה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המען בטבלת הסמלים.
8. עדכן  $IC \leftarrow IC + L$
9. חזור ל-2.
10. אם נמצאו שגיאות במעבר שני, עצור.
11. צור ושמור את קבצי הפלט: קובץ קוד המכונה קובץ סמלים חיצוניים, וקובץ סמלים של נקודות כניסה (ראו 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    r3,LENGTH
LOOP:      jmp     L1( #-1,r6)
           prn     #-5
           bne     LOOP(r4,r7)
           sub     r1, r4
           bne     END
L1:         inc     K
           bne     LOOP(K,STR)
END:        stop
STR:        .string "abcdef"
LENGTH:    .data   6,-9,15
K:          .data   22

```

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. את החלקים שעדיין לא מתורגמים במעבר זה, נשאיר כמות שהם.

אנו מניחים שהקוד ייטען לזיכרון החל מהמען 100 (בבסיס דצימאלי).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: mov r3,LENGTH	first word of instruction source register r3 address of label LENGTH	00000000110100 00001100000000 ?
0103 0104 0105 0106	LOOP: jmp L1( #-1,r6)	address of label L1 immediate #-1 (2's complement) immediate r6	00111001001000 ? 11111111111100 00000000011000
0107 0108	prn #-5	immediate #-5	00001100000000 11111111011100
0109 0110 0111	bne LOOP(r4,r5)	address of LOOP registers r4,r7	11111010001000 ? 00010000010100
0112 0113	sub r1, r4	registers r1,r4	00000011111100 00000100010000

0114 0115	bne     END	address of label END	00001010000100 ?
0116 0117	L1:     inc K	address of label K	00000111000100 ?
0118 0119 0120 0121	bne     LOOP(K,STR)	address of label LOOP address of label K address of label STR	01011010001000 ? ? ?
0122	END:    stop		00001111000000
0123	STR:    .string   "abcdef"	ascii code 'a'	00000001100001
0124		ascii code 'b'	00000001100010
0125		ascii code 'c'	00000001100011
0126		ascii code 'd'	00000001100100
0127		ascii code 'e'	00000001100101
0128		ascii code 'f'	00000001100110
0129		ascii code '\0' (end of string)	00000000000000
0130 0131 0132	LENGTH: .data 6,-9,15	integer 6 integer -9 (2's complement) integer 15	000000000000110 1111111110111 00000000001111
0133	K: .data 22	integer 22	00000000010110

טבלת הסמלים :

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	116
END	122
STR	123
LENGTH	130
K	133

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית :

Decimal Address	Source Code	Binary Machine Code
0100 0101 0102	MAIN: mov r3,LENGTH	00000000110100 00001100000000 00001000001010
0103 0104 0105 0106	LOOP: jmp L1(#-1,r6)	00111001001000 00000111010010 11111111111100 00000000011000
0107 0108	prn     #-5	00001100000000 11111111101100
0109 0110 0111	bne     LOOP(r4,r5)	11111010001000 00000110011110 00010000010100
0112 0113	sub     r1, r4	00000011111100 00000100010000
0114 0115	bne     END	00001010000100 00000111101010
0116 0117	L1:     inc K	00000111000100 00001000010110

0118	bne	LOOP(K,STR)	01011010001000
0119			00000110011110
0120			00001000010110
0121			00000111101110
0122	END:	stop	00001111000000
0123	STR:	.string "abcdef"	00000001100001
0124			00000001100010
0125			00000001100011
0126			00000001100100
0127			00000001100101
0128			00000001100110
0129			00000000000000
0130	LENGTH:	.data 6,-9,15	000000000000110
0131			11111111110111
0132			000000000001111
0133	K:	.data 22	00000000010110

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 לא יוצר את קובץ הפלט מסוג `externals`.  
אם אין בקובץ המקור אף הנחיית `entry`, האסמבלר לא יוצר 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 ללא ציון הסיומת.

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`



יווצר 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: 10 סיביות). 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 מאופסים.

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 וטיפוסו (external או relocatable).

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 מוצא). 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 מספר (מיעון מיידי) – 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, התו # מציין מיעון מיידי, תווית מציינת מיעון ישיר, שם של רגיסטר מציין מיעון רגיסטר, 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 לשתי הסיביות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכילו תמיד 0, מכיוון שאינן רלוונטיות לפעולה.

אם זוהי פעולה ללא אופרנדים (rts, stop) אזי תקודד רק המילה הראשונה (והיחידה). הסיביות של שיטות המיעון של שני 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, וסוג התווית הוא relocatable.

3. שורת הנחיה :

כאשר האסמבלר נתקל בהנחיה, 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. הטיפוס של התווית הוא relocatable, וכמו כן מסומן שההגדרה ניתנה בחלק הנתונים.

בסוף המעבר הראשון, ערך התווית יעודכן בטבלת הסמלים על ידי הוספת ה-IC (כלומר הוספת האורך הכולל של קידוד כל ההוראות). הסיבה לכך היא שבתמונת קוד המכונה, הנתונים מופרדים מההוראות, וכל הנתונים יופיעו אחרי כל ההוראות (ראו תאור קבצי הפלט בהמשך).

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 (כי גם האפס בסוף המחרוזת תופס מקום).

הטיפול בתווית המופיעה בשורה, 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 תווית ואם באופן עקיף על ידי הנחיית extern).

## פורמט קובץ ה-object

**האסמבלר בונה את תמונת זיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלי יכנס למען 100 (בבסיס 10) בזיכרון, קידוד ההוראה השנייה יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.**

מיד לאחר קידוד ההוראה האחרונה, מכניסים לתמונת הזיכרון את קידוד הנתונים שנבנו על ידי ההנחיות 'string'. 'data'. הנתונים יוכנסו בסדר בו הם מופיעים בקובץ המקור. אופרנד של

הוראה שמתייחס לסמל שהוגדר באותו קובץ, יקודד כך שיצביע על המקום המתאים בתמונת הזיכרון שבונה האסמבלר.

נשים לב שהמשתנים מופיעים בתמונת הזיכרון אחרי ההוראות. זוהי הסיבה בגללה יש לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המגדירים נתונים (סמלים מסוג .data).

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט כדלקמן:  
השורה הראשונה היא כותרת המכילה שני מספרים: האורך הכולל של קטע ההוראות (במילות זיכרון) ואחריו האורך הכולל של קטע הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.  
השורות הבאות מכילות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. כל המספרים בקובץ object הם בבסיס 2 **ייחודי** שהוגדר לעיל.  
קובץ object לדוגמה, כפי שאמור להיבנות על ידי האסמבלר, נמצא בהמשך.

### 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 2 **הייחודי** (ראו קובץ דוגמה בהמשך).  
אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

### 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. כמוכן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס 2 **הייחודי** (ראו קובץ דוגמה בהמשך).  
אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

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

נדגים את הפלט שמייצר האסמבלר עבור קובץ מקור בשם ps.as שהודגם קודם לכן.

התוכנית לאחר שלב פרישת המאקרו תיראה כך:

; file ps.as

```
.entry LENGTH
.extern W
MAIN:      mov    r3 ,LENGTH
LOOP:      jmp    L1(#-1,r6)
           prn    #-5
           bne    W(r4,r5)
           sub    r1, r4
           bne    L3
L1:         inc    K
.entry LOOP
           bne    LOOP(K,W)
END:       stop
STR:       .string "abcdef"
LENGTH:    .data   6,-9,15
K:         .data   22
.extern    L3
```

להלן טבלת הקידוד המלא הבינארי שמתקבל מקובץ המקור, ולאחריה הפורמטים קבצי הפלט השונים.

טבלת הקידוד הבינארי :

Decimal Address	Source Code	Binary Machine Code
0100 0101 0102	MAIN: mov r3 ,LENGTH	00000000110100 00001100000000 00001000001010
0103 0104 0105 0106	LOOP: jmp L1(#-1,r6)	00111001001000 00000111010010 11111111111100 00000000011000
0107 0108	prn #-5	00001100000000 11111111011100
0109 0110 0111	bne W(r4,r5)	11111010001000 00000000000001 00010000010100
0112 0113	sub r1, r4	00000011111100 00000100010000
0114 0115	bne L3	00001010000100 00000000000001
0116 0117	L1: inc K	00000111000100 00001000010110
0118 0119 0120 0121	bne LOOP(K,W)	01011010001000 00000110011110 00001000010110 00000000000001
0122	END: stop	00001111000000
0123	STR: .string "abcdef"	00000001100001
0124		00000001100010
0125		00000001100011
0126		00000001100100
0127		00000001100101
0128		00000001100110
0129		00000000000000
0130 0131 0132	LENGTH: .data 6,-9,15	00000000000110 11111111110111 00000000001111
0133	K: .data 22	00000000010110

הקובץ ps.ob :

כל תוכן הקובץ מיוצג במספרים בבסיס 2 הייחודי.

הערה 1: שורת הכותרת להלן אינה חלק מהקובץ, ונועדה להבהרה בלבד.

הערה 2: יש לקודד גם אפסים מובילים, כלומר 14 סיביות "ייחודיות" בכל מילה בקוד המכונה, ו-4 ספרות עשרוניות בכל כתובת.

Base 10 address    Base 2 code

	23	11
0100	.....	///..
0101	.....	///.....
0102	.....	///.....
0103	.....	///.....
0104	.....	///.....
0105	.....	////////..
0106	.....	///.....
0107	.....	///.....
0108	.....	////////..
0109	.....	///.....
0110	.....	///.....
0111	.....	///.....
0112	.....	////////..
0113	.....	///.....
0114	.....	///.....
0115	.....	///.....
0116	.....	///.....
0117	.....	///.....
0118	.....	///.....
0119	.....	///.....
0120	.....	///.....
0121	.....	///.....
0122	.....	///.....
0123	.....	///.....
0124	.....	///.....
0125	.....	///.....
0126	.....	///.....
0127	.....	///.....
0128	.....	///.....
0129	.....	///.....
0130	.....	///.....
0131	.....	////////..
0132	.....	///.....
0133	.....	///.....

#### הקובץ ps.ent:

כל ערכי הסמלים מיוצגים בבסיס 10.

LOOP	103
LENGTH	130

#### הקובץ ps.ext:

כל הכתובות מיוצגות בבסיס 10.

W	110
L3	115
W	121

לתשומת לב: אם בקובץ המקור אין הצהרת extern. אזי לא ייווצר עבורו קובץ ext. בדומה, אם אין בקובץ המקור הצהרת 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 לאחסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקור), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיימות. 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, ממשיך המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) 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 כל אחד מהמנחים בשעות הקבלה שלהם.**

להזכירכם, באפשרותו של כל סטודנט לפנות לכל מנחה, לאו דווקא למנחה הקבוצה שלו, לקבלת עזרה. שוב מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר זה הרבה שאלות בנושא חומר הלימוד והממ"נים, והתשובות יכולות להועיל לכולם.

לתשומת לבכם: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

**ב ה צ ל ח ה !**