

האוניברסיטה הפתוחה

20465

## **מעבדה בתכנות מערכות**

חוברת הקורס – אביב 2024ב

כתבה: מיכל אבימור

מרץ 2024 – סמסטר אביב – תשפ"ד

**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
17	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 הספרייה 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** (קורס 20465 / ב2024)

שבוע לימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח ממ"ן (למנחה)
1	22.03.2024-17.03.2024 (ה תענית אסתר)	ספר C פרקים 1-2-3	מפגש ראשון	
2	29.03.2024-24.03.2024 (א פורים)	ספר C פרקים 1-2-3		
3	05.04.2024-31.03.2024	ספר C פרק 4	מפגש שני	
4	12.04.2024-07.04.2024	ספר C פרק 4		
5	19.04.2024-14.04.2024	ספר C פרק 5	מפגש שלישי	ממ"ן 11 14.04.2024
6	26.04.2024-21.04.2024 (ב-ו פסח)	ספר C פרק 5		
7	03.05.2024-28.04.2024 (א-ב פסח)	ספר C פרק 6	מפגש רביעי	
8	10.05.2024-05.05.2024 (ב יום הזכרון לשואה)	ספר C פרק 6		
9	17.05.2024-12.05.2024 (ב יום הזיכרון, ג יום העצמאות)	ספר C פרק 6,7	מפגש חמישי	ממ"ן 12 12.05.2024

\* 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	24.05.2024-19.05.2024	ספר C פרק 7	מפגש שישי	
11	31.05.2024-26.05.2024 (א ל"ג בעומר)	ספר C פרק 7 + פרויקט		
12	07.06.2024-02.06.2024	פרויקט וחזרה	מפגש שביעי	ממ"ן 22 02.06.2024
13	14.06.2024-09.06.2024 (ד שבועות)	פרויקט וחזרה		
14	21.06.2024-16.06.2024	פרויקט וחזרה		ממ"ן 23 16.06.2024  ממ"ן 14** 11.08.2024

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 במיוחד, 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:

- הערה בראש תכנית שתכלול תיאור תמציתי של מטרות התכנית, כיצד מושגת מטרה זו, תיאור המודלים והאלגוריתם, קלט/פלט וכל הנחה שהנכם מניחים.
- 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.
- 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: 2024 מועד אחרון להגשה: 14.04.2024

אופן ההגשת המטלות:

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

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

הקבצים של כל תוכנית יהיו בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה, ומאפשרות בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית. את המטלה יש להגיש בקובץ zip.

חשוב מאוד:

- אופן הגשת המטלה והקבצים הנדרשים להגשה מופיעים כאן וכן בעמודים ה-ז בסעיף תיאור המטלות. במקרה של הנחיה 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 my\_add.c) (50 נקודות)

עליכם לכתוב תכנית המכילה פונקציה:

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

המבצעת סימולציה של חיבור בבסיס 2, בין שני האופרנדים a, b.

לדוגמה, אם a הוא 42, ובייצוג בינארי 000000000101010

ואם b הוא 123, ובייצוג בינארי 0000000001111011

אז על הפונקציה לחבר שני מספרים אלה באופן הבא:

תחילה, מחברים את שתי הספרות הבינאריות הימניות ביותר. תוצאת חיבור זה היא  $1=0+1$ , ואין נשא. עוברים לזוג הספרות הבינאריות השני מימין. כאן תוצאת החיבור היא  $10=1+1$  (בבסיס 2), כלומר הספרה 0 והנשא 1. את הנשא מעבירים לטור הבא, ומחברים אותו עם זוג הספרות הבינאריות השלישי מימין, וכך הלאה עד הקצה השמאלי של המספרים.

על הפונקציה להדפיס באופן נאה בבסיס 2 את האופרנדים ואת תוצאת החיבור. כמו כן, הפונקציה תחזיר את תוצאת החיבור כמספר מטיפוס `unsigned int`.

לתשומת לב: אין לבצע חיבור "רגיל" של המשתנים `a`, `b`, אלא לממש חיבור פרטני בבסיס 2 כמתואר לעיל.

רמז: אפשר לגשת לכל סיבית של מספר מטיפוס `unsigned int` באמצעות פעולות מתמטיות מגוונות של שפת C.

בנוסף, עליכם לכתוב תכנית ראשית (הפונקציה `main`), שתבצע קלט של שני האופרנדים, ותפעיל את הפונקציה `my_add` על נתוני הקלט. כמו כן, לאחר החזרה מהפונקציה, התכנית הראשית תדפיס באופן נאה בבסיס 10 את האופרנדים, ואת הסכום שהוחזר מהפונקציה.

נתוני הקלט הם שני מספרים שלמים אי-שליליים בבסיס 10, כל מספר ברוחב של עד 6 ספרות עשרוניות משמעותיות. מותר להניח שהקלט תקין.

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 הרצות בדיקה (אחת או יותר), המדגימות את פעולת התכנית על מגוון קלטים. יש להגיש תדפיסי מסך של כל ההרצות וכן את קבצי הקלט.

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

עליכם לכתוב פונקציה `int suffix(char str[], char c)`. פונקציה זו מדפיסה את כל הסייפות של המחרוזת `str` אשר מתחילות בתו `c`. כמו כן על הפונקציה להחזיר את מספר הסייפות שמצאה.

כל סייפא תודפס בשורה נפרדת. אם התו `c` לא נמצא במחרוזת `str`, הפונקציה לא תדפיס דבר, ותחזיר 0.

לדוגמה: הקריאה `suffix("abrahmrnsbre", 'b')` תדפיס:

```
brahmrnsbre
bre
```

ותחזיר 2.

בנוסף, עליכם לכתוב תכנית ראשית (הפונקציה `main`) שתקרא מהקלט תו בודד `c` ולאחריו מחרוזת תווים. התכנית הראשית תקרא לפונקציה `suffix` עם נתוני הקלט, ולאחר החזרה מהפונקציה תדפיס הודעה נאה שמציינת את מספר הסייפות שהפונקציה מצאה.

המחרוזת המועברת בקלט מסתיימת בסוף שורת הקלט, ותהיה באורך מקסימלי של 127 תווים (לא כולל ה-0 המסיים). הניחו שהקלט תקין. אין צורך לטפל בשגיאות בקלט.  
**לתשומת לב:** אין לכלול במחרוזת את התו `\n` (שורה חדשה) שמופיע בסוף שורת קלט.

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 יזום מתוך התכנית 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 (כולל מקרי קצה). 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 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: 2024' מועד אחרון להגשה: 12.05.2024

## אופן הגשת המטלות:

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

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: `Wall-ansi-pedantic`. יש להגיש את קבצי המקור `.c` (`.h` 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).

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 האו"פ למחשב האישי, ולבדוק שהקבצים אכן הועלו למערכת באופן תקין.

## שאלה 1 (תכנית ראשית בקובץ `partialSums.c`)

עליכם לכתוב פונקציה בשם `partialSums`, אשר מקבלת כפרמטרים מערך של מספרים שלמים מטיפוס `int`, ואת גודל המערך. הפונקציה מחשבת ומחזירה מערך של "סכומים חלקיים", שבו כל איבר מהמערך המקורי מוחלף בסכום האיברים עד אליו (כולל) החל מאינדקס 0. **אסור** לפונקציה לגרום לשינויים במערך המקורי (רמז: חישבו כיצד להחזיר את מערך הסכומים).

## לדוגמה:

בהינתן המערך הבא בגודל 8 (משמאל לימין):

14 10 -1 10 -15 1 5 3

הפונקציה תחזיר את המערך הבא:

27 13 3 -6 9 8 3

כמו כן עליכם לכתוב תכנית ראשית (הפונקציה main) הקולטת מהמשתמש גודל של מערך, ואז קולטת מהמשתמש את איברי המערך (מספרים שלמים). התכנית הראשית קוראת לפונקציה partialSums עם מערך הקלט, ואחרי החזרה מהפונקציה מדפיסה באופן נאה את מערך הסכומים החלקיים.

בקלט ובפלט יש להשתמש בייצוג עשרוני. הניחו כי הקלט תקין. אין צורך לטפל בשגיאות בקלט.

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

על התוכנית להדפיס הודעת בקשה ידידותית בכל פעם כשנדרש קלט.

כמו כן יש להדפיס באופן יזום מתוך התוכנית את הנתונים כפי שנקלטו, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג במסך גם כאשר הוא מגיע מקובץ (כידוע, נתונים הנקראים מקובץ אינם מוצגים במסך בזמן הקלט).

חובה לצרף להגשה מספר הרצות דוגמה, המדגימות את פעולת התוכנית על מערכים בגדלים שונים ועל מגוון ערכים בכל מערך. 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 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: 2024ב' מועד אחרון להגשה: 02.06.2024

## אופן הגשת המטלות:

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

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: `-Wall -ansi -pedantic`. יש להגיש את קבצי המקור `.c` (`.h` 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).

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`.

## חשוב מאוד:

- אופן הגשת המטלה והקבצים הנדרשים להגשה מופיעים כאן וכן בעמודים ה-ז בסעיף תיאור המטלות. במקרה של הנחיה 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 `myset.c`, ובנוסף הקבצים העיקריים `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  
Multiple consecutive commas  
יש להגיב בהודעה כגון :
14. לפקודה :  
intersect\_set SETF SETD SETA  
Missing comma  
יש להגיב בהודעה כגון :
15. לפקודה :  
syndiff\_set, SETF, SETB, SETA  
Illegal comma  
יש להגיב בהודעה כגון :

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: read\_set , print\_set , union\_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 לדיבוג התכנית. בכל קובץ קלט תהיה סדרה של פקודות על קבוצות.
- על התכנית להדפיס הודעת בקשה ידיוותית 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 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 6,7,8

משקל המטלה: 12 נקודות (רשות)

מספר השאלות: 2

מועד אחרון להגשה: 16.06.2024

סמסטר: 2024ב'

## 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. 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). קבצי התכנית 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.

## 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)

עליכם להסביר במילים את משמעותו של הביטוי הבא :

```
int *(*p)[2][2](int,int);
```

יש להסביר את הביטוי בדיוקנות ובאופן מלא. ניתן להוסיף דוגמה, לצורך הבהרה.

## לתשומת לב:

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

עליכם לכתוב תכנית המקבלת, באמצעות ארגומנטים בשורת הפקודה, רשימת שמות של קבצים.

כל קובץ כזה מכיל סדרת מספרים שלמים בבסיס עשרוני, הלקוחים מהתחום הסגור [0..28]. המספרים אינם מופיעים בסדר מוגדר כלשהו, וייתכנו מופעים מרובים של אותו מספר. בין כל שני מספרים סמוכים מפרידים תווים לבנים (אחד או יותר).

על התוכנית לקרוא את המספרים מכל הקבצים. בגמר הקלט, התכנית תדפיס לפלט הסטנדרטי שורת סיכום עבור כל מספר המופיע באחד או יותר מקבצי הקלט. הסיכום מפרט את שמות כל הקבצים בהם המספר מופיע, וכמה פעמים הוא מופיע בכל קובץ. עבור מספר שאינו מופיע באף קובץ, לא יודפס דבר.

### דוגמה:

נתונים שלשת הקבצים הבאים בתיקיה של התכנית hash

הקובץ dat1.in : 11 2 4 21 23 21

הקובץ dat2.in : 21 7 4 23 21 13 13

הקובץ dat3.in : 5 23 5 23 6 23 23

נריץ את שורת הפקודה : ./hash dat1.in dat2.in dat3.in

יתקבל הפלט הבא :

2 appears in file dat1.in - 1 time

4 appears in file dat1.in - 1 time, file dat2.in - 1 time

5 appears in file dat3.in - 2 times

6 appears in file dat3.in - 1 time

7 appears in file dat2.in - 1 time

11 appears in file dat1.in - 1 time

13 appears in file dat2.in - 2 times

21 appears in file dat1.in - 2 times, file dat2.in - 2 times

23 appears in file dat1.in - 1 time, file dat2.in - 1 time, file dat3.in - 4 times

על מנת לבנות פתרון יעיל, נרצה להשתמש במבנה נתונים חסכוני מבחינת גודל הזיכרון הנדרש.

נשים לב שתחום הערכים אמנם חסום ולא גדול, אבל מספר המופעים של כל ערך בכל קובץ אינו מוגבל, וגם מספר הקבצים אינו מוגבל.

עליכם להגדיר מבנה נתונים דינמי, הנקרא טבלת-גיבוב (hash-table). טבלת-הגיבוב מורכבת מ-29 רשימות מקושרות, אחת לכל ערך שלם בתחום [0..28]. ברשימה המקושרת של הערך k, כל איבר מכיל פרטים על כמות המופעים של k באחד מקבצי הקלט. לפיכך, האורך המקסימלי של הרשימה המקושרת הוא כמספר הקבצים בהם k מופיע. בתחילת התכנית, כל הרשימות המקושרות ריקות.

#### **דרישות מיוחדות לביצוע הקלט:**

- אסור להחזיק יותר מקובץ קלט אחד פתוח בכל רגע נתון.
- מותר לפתוח כל קובץ קלט פעם אחת בלבד במהלך התכנית.
- מותר לקרוא כל נתון מהקובץ פעם אחת בלבד.

על התוכנית להדפיס הודעת שגיאה לקובץ השגיאות הסטנדרטי ולעצור את עבודתה, במקרה שאחד מקבצי הקלט אינו ניתן לפתיחה, או אם לא הועברו בכלל ארגומנטים בשורת הפקודה.

ניתן להניח (אין צורך לבדוק) שתוכן הקבצים בפורמט תקין, ושהערכים בתחום המוגדר, כמפורט לעיל.

**חובה לצרף להגשה מספר הרצות בדיקה (לפחות שתי הרצות תקינות, ואחת עם הודעת שגיאה), המדגימות את פעולת התכנית על קבצי קלט מגוונים וערכים מגוונים בכל קובץ. יש להגיש את קבצי הקלט, וכן תדפיסי מסך (או קבצי פלט) של כל ההרצות.**

**לתשומת לבכם:** 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 : 2024 מועד אחרון להגשה : 11.08.2024

## 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 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 ותייעוד 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 נוה 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 המושג

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), 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 15 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 14. 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 4096 תאים, בכתובות 0-4095 (בבסיס עשרוני), וכל תא הוא בגודל של 15 סיביות. לתא בזיכרון נקרא גם בשם **"מילה"**. הסיביות בכל מילה ממוספרות כמו ברגיסטר.

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

### מבנה הוראת מכונה:

כל הוראת מכונה מקודדת למספר מילות זיכרון רצופות, **החל ממילה אחת ועד למקסימום שלוש מילים**, בהתאם לשיטות המיעון בהן נעשה שימוש (ראו בהמשך).

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

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

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				שיטת מיעון אופרנד מקור				שיטת מיעון אופרנד יעד				השדה A,R,E		
opcode				ש	ש	ש	ש	ש	ש	ש	ש	A	R	E
				י	י	י	י	י	י	י	י			
				ט	ט	ט	ט	ט	ט	ט	ט			
				ה	ה	ה	ה	ה	ה	ה	ה			
				3	2	1	0	3	2	1	0			

פירוט השיטות 0-3 מופיע בסעיף "שיטת מיעון" בהמשך.

בשפה שלנו יש 16 קודי פעולה והם :

שם הפעולה	קוד הפעולה (בבסיס עשרוני)
mov	0
cmp	1
add	2
sub	3
lea	4
clr	5
not	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 11-14:** במילה הראשונה של ההוראה סיביות אלה מהוות את **קוד-הפעולה** (opcode). כל opcode מיוצג בשפת אסמבלי באופן סימבולי על ידי **שם-פעולה**.

**סיביות 7-10:** מקודדות את שיטת המיעון של אופרנד המקור (source operand). לכל שיטת מיעון יש סיבית נפרדת, שערכה 1 אם אופרנד המקור נתון בשיטה זו, ואחרת ערך הסיבית 0. אם אין בהוראה אופרנד מקור, כל ארבע הסיביות מאופסות.

**סיביות 3-6:** מקודדות את שיטת המיעון של אופרנד היעד (destination operand). לכל שיטת מיעון יש סיבית נפרדת, שערכה 1 אם אופרנד היעד נתון בשיטה זו, ואחרת ערך הסיבית 0. אם אין בהוראה אופרנד יעד, כל ארבע הסיביות מאופסות.

**סיביות 0-2 (השדה 'A,R,E'):** אפיון של תפקיד השדה 'A,R,E' בקוד המכונה יובא בהמשך. במילה הראשונה של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות מאופסות.

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

#### שיטות מיעון:

בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3. השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בקוד המכונה של כל הוראת מכונה. כל אופרנד של ההוראה דורש מילה אחת נוספת.



כאשר בהוראה יש שני אופרנדים, קודם תופיע מילת-המידע הנוספת של האופרנד הראשון (אופרנד המקור), ולאחריה מילת-המידע הנוספת של האופרנד השני (אופרנד היעד). קיים גם מקרה מיוחד בו קידוד שני האופרנדים נעשה באמצעות מילת-מידע אחת משותפת לשני האופרנדים.

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

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

הסבר על התפקיד של השדה 'A,R,E' בקוד המכונה יבוא בהמשך.

ערך השדה 'A,R,E' הנדרש בכל שיטת מיעון מופיע בתיאור שיטות המיעון להלן.

ערך	שיטת המיעון	תוכן המילה הנוספת	אופן הכתיבה	דוגמה
0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 12 סיביות, השוכן בסיביות 14-3 של המילה.  הסיביות 2-0 של מילת המידע הן השדה A,R,E. במיעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1,r2  בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2
1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר ללא סימן ברוחב של 12 סיביות, בסיביות 14-3 של מילת המידע.  הסיביות 2-0 במילת המידע הן השדה A,R,E. במיעון ישיר, ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 14-3. אם זוהי כתובת שמייצגת שורה בקובץ המקור הנוכחי (כתובת פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זוהי כתובת שמייצגת שורה בקובץ מקור אחר של התכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת תווית בתחילת הנחית 'data' או 'string', או בתחילת הוראה של התוכנית, או באמצעות אופרנד של הנחית 'extern'.  תווית היא ייצוג סימבולי של כתובת בזיכרון.	השורה הבאה מגדירה את התווית x:  x: .data 23  ההוראה: dec x  מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x).

ערך	שיטת המיעון	תוכן המילה הנוספת	אופן הכתיבה	דוגמה
2	מיעון אוגר עקיף	<p>שיטת מיעון זו משמשת לגישה לזיכרון באמצעות מצביע שנמצא באוגר. תוכן האוגר הוא כתובת בזיכרון, והמילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת באוגר כמספר ללא סימן ברוחב של 15 סיביות.</p> <p>אם האופרנד הוא אופרנד יעד, מילת-מידע נוספת של הפקודה תכיל בסיביות 3-5 את מספרו של האוגר שמשמש כמצביע. ואילו אם האוגר הוא אופרנד מקור, מספר האוגר יקודד בסיביות 6-8 של מילת-המידע הנוספת.</p> <p>הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון אוגר עקיף, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p> <p>אם בפקודה יש שני אופרנדים, וכל אחד מהם בשיטת מיעון אוגר עקיף או בשיטת מיעון אוגר ישיר (ראו בהמשך), שני האוגרים יחלקו מילת-מידע אחת משותפת, כאשר סיביות 3-5 יכילו את מספר אוגר היעד, וסיביות 6-8 את מספר אוגר המקור. השדה A,R,E יהיה כמפורט לעיל.</p> <p>סיביות במילת-המידע שאינן בשימוש יכילו 0.</p>	<p>האופרנד מתחיל בתו * ולאחריו ובצמוד אליו מופיע שם של אוגר.</p>	<p>inc *r1</p> <p>בדוגמה זו, ההוראה inc מגדילה ב-1 את תוכן המילה בזיכרון עליה מצביע האוגר r1, כלומר המילה שכתובתה נמצאת באוגר r1.</p> <p>דוגמה נוספת:</p> <p>mov *r1,*r2</p> <p>בדוגמה זו, ההוראה mov מעתיקה את תוכן המילה בזיכרון עליה מצביע האוגר r1 אל המילה בזיכרון עליה מצביע האוגר r2.</p> <p>שני האופרנדים בדוגמה זו הם בשיטת מיעון אוגר עקיף, ולכן שני האוגרים יקודדו במילת-מידע נוספת אחת משותפת.</p>
3	מיעון אוגר ישיר	<p>האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילת-מידע נוספת של הפקודה תכיל בסיביות 3-5 את מספרו של האוגר. ואילו אם האוגר משמש כאופרנד מקור, מספר האוגר יקודד בסיביות 6-8 של מילת-המידע.</p> <p>הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון אוגר ישיר, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p> <p>אם בפקודה יש שני אופרנדים, וכל אחד מהם בשיטת מיעון אוגר ישיר או בשיטת מיעון אוגר עקיף (ראו לעיל), שני האוגרים יחלקו מילת-מידע אחת משותפת, כאשר סיביות 3-5 יכילו את מספר אוגר היעד, וסיביות 6-8 את מספר אוגר המקור. השדה A,R,E יהיה כמפורט לעיל.</p> <p>סיביות במילת-המידע שאינן בשימוש יכילו 0.</p>	האופרנד הוא שם של אוגר.	<p>clr r1</p> <p>בדוגמה זו, ההוראה clr מאפסת את תוכן האוגר r1.</p> <p>דוגמה נוספת:</p> <p>mov r1,*r2</p> <p>בדוגמה זו, ההוראה mov מעתיקה את תוכן האוגר r1 אל המילה בזיכרון אליה מצביע האוגר r2.</p> <p>אופרנד המקור r1 נתון בשיטת מיעון אוגר ישיר, ואופרנד היעד *r2 בשיטת מיעון אוגר עקיף, ולכן שני האוגרים יקודדו במילת-מידע נוספת אחת משותפת.</p>

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 רגיסטר פנימי של המעבד (לא רגיסטר כללי), שמכיל בכל רגע נתון את כתובת הזיכרון בה נמצאת ההוראה הנוכחית שמתבצעת (הכוונה תמיד לכתובת המילה הראשונה של ההוראה).

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

#### קבוצת ההוראות הראשונה:

אלו הן הוראות המקבלות שני אופרנדים.

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

הוראה	הסבר הפעולה	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 את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל רגיסטר 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 r0.
sub	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	רגיסטר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	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 השנייה:

אלו הן הוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. במקרה זה, השדה של אופרנד המקור (סיביות 7-10) במילה הראשונה בקידוד ההוראה הינו חסר משמעות, ולפיכך יכול אפסים.

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

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
clr	איפוס תוכן האופרנד	clr r2	$r2 \leftarrow 0$
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	$r2 \leftarrow \text{not } r2$
inc	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
dec	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$
jmp	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp LINE	$PC \leftarrow \text{LINE}$ מציביע התוכנית מקבל את המען המיוצג על ידי התווית LINE, ולפיכך הפקודה הבאה שתבצע תהיה במען זה.
bne	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת cmp.	bne LINE	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי: $PC \leftarrow \text{LINE}$
red	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לפלט הסטנדרטי.
jsr	קריאה לשגרה (סברוטינה), מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזיכרון המחשב, והאופרנד מוכנס ל-PC.	jsr FUNC	push(PC) $PC \leftarrow \text{FUNC}$

### קבוצת ההוראות השלישית:

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

ההוראות השייכות לקבוצה זו הן: `rts`, `stop`.

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
<code>rts</code>	חזרה משיגרה. הערך שנמצא בראש המחשנית של המחשב מוצא מן המחשנית, ומוכנס אל מצביע התוכנית ( <code>PC</code> ).	<code>rts</code>	<code>PC ← pop()</code>
<code>stop</code>	עצירת ריצת התוכנית.	<code>stop</code>	התכנית עוצרת

### מבנה שפת האסמבלי:

תכנית בשפת אסמבלי בנויה ממאקרואים וממשפטים (`statements`).

### מאקרואים:

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 `m_macr`)

```
macr m_macr
  inc r2
  mov A,r1
endmacr
```

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

```
.
.
.
m_macr
.
.
m_macr
.
.
.
```

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

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

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

### הנחות והנחיות לגבי מאקרו:

- 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 `endmacro` (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.

### לסיכום, במאקרו יש לבדוק:

- (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.

### 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 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.

קעת נפרט יותר 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: למילים בקוד המכונה הנוצרות ממשפט הנחיה 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 התו '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה למה שנעשה עבור 'data'. (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחרוזת).

לדוגמה, ההנחיה:

STR: .string "abcdef"

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

3. ההנחיה 'entry'.

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

לדוגמה, השורות:

```
HELLO    .entry
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 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

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 הנחית data או string. תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

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, 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 האסמבלר.

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

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

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

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

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

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

```
MAIN:      add    r3, LIST
LOOP:      prn    #48
           macr   m_macr
           cmp    r3, #-6
           bne    END
           endmacr
           lea    STR, r6
           inc    r6
           mov    *r6, K
           sub    r1, r4
           m_macr
```

```

                dec    K
                jmp    LOOP
END:            stop
STR:            .string "abcd"
LIST:           .data  6, -9
                .data  -100
K:              .data  31

```

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

```

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

```

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

### אלגוריתם שלדי של קדם האסמבלר

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

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עבור ל- 9 (סיום).
2. האם השדה הראשון הוא שם מאקרו המופיע בטבלת המאקרו (כגון `m_macro`)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא "`macro`" (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "`macro`".
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה `m_macro`).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום). אם דגל "`macro`" דולק ולא זוהתה תווית `endmacro` הכנס את השורה לטבלת המאקרו ומחק את השורה הנ"ל מהקובץ. אחרת (לא מאקרו) חזור ל- 1.
7. האם זוהתה תווית `endmacro`? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל "`macro`". 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 mov, jmp, prn, sub, cmp, inc, bne, stop בקוד הבינארי השקול להם במודל המחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים K,STR, LIST, MAIN, LOOP, END 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:

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010000010010
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 000001111101010 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6,K	Source register 6 Address of label K	000001000010100 000000110000100 000010000101010
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 11111111010100
0118 0119	bne END	Address of label END	101000000010100 000001111001010
0120 0121	dec K	Address of label K	011100000010100 000010000001010
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 000001100111010
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0129		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111101111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

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 124 (עשרוני), ושהסמל K משויך למען 133, 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 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 לעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	103
END	124
STR	125
LIST	130
K	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 המשוך לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

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

## המעבר השני

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

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

## הפרדת הוראות ונתונים

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

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

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

## גילוי שגיאות בתכנית המקור

הנחת המטלה היא שאין שגיאות בהגדרות המאקרו, ולכן שלב קדם האסמבלר אינו מכיל שלב גילוי שגיאות, 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 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 של שיטות המיעון החוקיות, עבור אופרנד המקור ואופרנד היעד של ההוראות השונות הקיימות בשפה הנתונה:

שם פעולה	שיטות מיעון חוקיות עבור אופרנד מקור	שיטות מיעון חוקיות עבור אופרנד יעד
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
lea	1	1,2,3
clr	אין אופרנד מקור	1,2,3
not	אין אופרנד מקור	1,2,3
inc	אין אופרנד מקור	1,2,3
dec	אין אופרנד מקור	1,2,3
jmp	אין אופרנד מקור	1,2
bne	אין אופרנד מקור	1,2
red	אין אופרנד מקור	1,2,3
prn	אין אופרנד מקור	0,1,2,3
jsr	אין אופרנד מקור	1,2
rts	אין אופרנד מקור	אין אופרנד יעד
stop	אין אופרנד מקור	אין אופרנד יעד

### אלגוריתם שלדי של האסמבלר

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

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

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 בשורה הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data או string? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין data. ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, ועדכן את מונה הנתונים DC בהתאם לאורכם. חזור ל-2.
8. האם זו הנחית extern או הנחית entry? אם לא, עבור ל-11.
9. האם זוהי הנחית extern? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם המאפיין external. חזור ל-2.
10. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו יהיה  $IC+100$  (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
11. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודע על שגיאה בשם ההוראה.
12. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה.
13. עדכן  $IC \leftarrow IC + L$ , וחזור ל-2.
14. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
15. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין כ- data, ע"י הוספת הערך  $IC+100$  (ראה הסבר בהמשך).
16. התחל מעבר שני.

### מעבר שני

1. אתחל  $IC \leftarrow 0$ .
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-9.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הנחית data או string או extern? אם כן, חזור ל-2.
5. האם זוהי הנחית entry? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים במאפיין entry. חזור ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השנייה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד מכיל סמל, מצא את הערך בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה).
8. עדכן  $IC \leftarrow IC + L$ , וחזור ל-2.
9. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
10. בנה את קבצי הפלט (פרטים נוספים בהמשך).

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

```
.MAIN:      add     r3, LIST
LOOP:      prn     #48
           lea     STR, r6
           inc     r6
           mov     *r6, K
           sub     r1, r4
           cmp     r3, #-6
           bne     END
           dec     K
```



```

        jmp    LOOP
END:     stop
STR:     .string "abcd"
LIST:    .data  6, -9
        .data  -100
K:       .data  31

```

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

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

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 ?
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 ? 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6,K	Source register 6 Address of label K	000001000010100 000000110000100 ?
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 11111111010100
0118 0119	bne END	Address of label END	101000000010100 ?
0120 0121	dec K	Address of label K	011100000010100 ?
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 ?
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0125		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111101111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

טבלת הסמלים :

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	103
END	124
STR	125
LIST	130
K	133

נבצע עתה את המעבר השני. נשלים את הקידוד החסר באמצעות טבלת הסמלים, ונרשום את הקוד בצורתו הסופית :

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010000010010
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 000001111101010 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6, K	Source register 6 Address of label K	000001000010100 000000110000100 000010000101010
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 111111111010100
0118 0119	bne END	Address of label END	101000000010100 000001111001010
0120 0121	dec K	Address of label K	011100000010100 000010000001010
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 000001100111010
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0125		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111110111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

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`  
ייוצר קובץ פלט `x.ob`, וכן קבצי פלט `x.ent` ו-`x.ext` 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 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,

ומאפיינים שונים שצוינו קודם, כגון המיקום (code או data), או אופן העדכון (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 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 "משריין" 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 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.

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. הטיפוס של התווית הוא relocatable, וכמו כן מסומן שהגדרה 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 תיאור 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 (גם האפס בסוף המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בשורה זו זהה לטיפול הנעשה בהנחיה 'data'.

## III. 'entry'.

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

## IV. 'extern'.

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

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

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

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

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

## פורמט קובץ ה-object

האסמבלר בונה את תמונת זיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלי ייכנס למען 100 (בבסיס עשרוני) 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 בקובץ ה-object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): האורך הכולל של קטע ההוראות (במילות זיכרון) ואחריה האורך הכולל של קטע הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה שני ערכים: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס עשרוני בארבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם בבסיס אוקטאלי ב-5 ספרות (כולל אפסים מובילים). בין שני הערכים בכל שורה יפריד רווח אחד.

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 עשרוני.

## פורמט קובץ ה-externals

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

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

```

; file ps.as

.entry LIST
.extern fn1
MAIN:      add    r3, LIST
           jsr    fn1
LOOP:      prn    #48
           lea    STR, r6
           inc    r6
           mov    *r6, L3
           sub    r1, r4
           cmp    r3, #-6
           bne    END
           add    r7, *r6
           clr    K
           sub    L3, L3

.entry MAIN
           jmp    LOOP
END:       stop
STR:       .string "abcd"
LIST:      .data  6, -9
           .data  -100
K:         .data  31
.extern L3

```

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

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010001001010
0103 0104	jsr fn1	Address of label fn1 (external)	110100000010100 000000000000001
0105 0106	LOOP: prn #48	Immediate value 48	110000000001100 000000110000100
0107 0108 0109	lea STR, r6	Address of label STR Target register 6	010000101000100 000010000100010 000000000110100
0110 0111	inc r6	Target register 6	011100001000100 000000000110100
0112 0113 0114	mov *r6, L3	Source register 6 Address of label L3 (external)	000001000010100 000000110000100 000000000000001
0115 0116	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0117 0118 0119	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 111111110101010
0120 0121	bne END	Address of label END	101000000010100 000010000011010
0122 0123	add r7, *r6	Source register r0 and target register 6	001010000100100 000000111110100
0124 0125	clr K	Address of label K	010100000010100 000010001100010
0126 0127 0128	sub L3, L3	Address of label L3 (external) Address of label L3 (external)	001100100010100 000000000000001 000000000000001
0129 0130	jmp LOOP	Address of label LOOP	100100000010100 000001101001010
0131	END: stop		111100000000100
0132	STR: .string "abcd"	Ascii code 'a'	000000001100001
0133		Ascii code 'b'	000000001100010
0134		Ascii code 'c'	000000001100011
0135		Ascii code 'd'	000000001100100
0136		Ascii code '\0' (end of string)	000000000000000
0137 0138	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111110111
0139	.data -100	Integer -100	111111110011100
0140	K: .data 31	Integer 31	000000000011111



להלן תוכן קבצי הפלט של הדוגמה. הערה: יש חשיבות לעימוד הנתונים בקבצים  
הקובץ ps.ob:

```

32 9
0100 12024
0101 00304
0102 02112
0103 64024
0104 00001
0105 60014
0106 00604
0107 20504
0108 02042
0109 00064
0110 34104
0111 00064
0112 01024
0113 00604
0114 00001
0115 16104
0116 00144
0117 06014
0118 00304
0119 77724
0120 50024
0121 02032
0122 12044
0123 00764
0124 24024
0125 02142
0126 14424
0127 00001
0128 00001
0129 44024
0130 01512
0131 74004
0132 00141
0133 00142
0134 00143
0135 00144
0136 00000
0137 00006
0138 77767
0139 77634
0140 00037

```

הקובץ ps.ent:

כל המספרים בבסיס עשרוני

```

MAIN 100
LIST 137

```

הקובץ ps.ext:

כל המספרים בבסיס עשרוני

```

fn1 0104
L3 0114
L3 0127
L3 0128

```

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 לאחסון 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, ממשיק 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 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 חריגים במיוחד. במקרים אלו יש לבקש ולקבל אישור מראש ממנחה הקבוצה.

**ב ה צ ל ח ה !**