ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – אביב 2020ב

כתבה: מיכל אבימור

מרץ 2020 – סמסטר אביב – תש״פ

פנימי – לא להפצה.

תוכן העניינים

אל הסטודנט,

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

לקורס קיים אתר באינטרנט בו תמצאו חומרי למידה נוספים, אותם מפרסם/מת מרכז/ת ההוראה. בנוסף, האתר מהווה עבורכם ערוץ תקשורת עם צוות ההוראה ועם סטודנטים אחרים בקורס.
פרטים על למידה מתוקשבת ואתר הקורס, תמצאו באתר שה"ם בכתובת:
http://telem.openu.ac.il
מידע על שירותי ספרייה ומקורות מידע שהאוניברסיטה מעמידה לרשותכם, תמצאו באתר הספריה באינטרנט www.openu.ac.il/Library.

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

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

לתשומת לב הסטודנטים הלומדים בחו"ל:

למרות הריחוק הפיסי הגדול, נשתדל לשמור אתכם על קשרים הדוקים ולעמוד לרשותכם ככל האפשר.
הפרטים החיוניים על הקורס נכללים בחוברת הקורס וכן באתר הקורס.
מומלץ מאד להשתמש באתר הקורס ובכל אמצעי העזר שבו וכמובן לפנות אלינו במידת הצורך.

אני מאחלת לך לימוד פורה ומהנה.

בברכה,

מיכל אבימור
מרכזת ההוראה בקורס.

1. לוח זמנים ופעילויות (20465 / ב2020)

שבוע לימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח ממ״ן (למנחה)
1	20.03.2020-15.03.2020	ספר C פרקים 3-2-1	מפגש ראשון	
2	27.03.2020-22.03.2020	ספר C פרקים 3-2-1		
3	03.04.2020-29.03.2020	ספר C פרק 4	מפגש שני	
4	10.04.2020-05.04.2020 (ד ערב פסח) (ה-ו פסח)	ספר C פרק 4		
5	17.04.2020-12.04.2020 (א-ד פסח)	ספר C פרק 5	מפגש שלישי	ממ״ן 11 12.04.2020
6	24.04.2020-19.04.2020 (ג יום הזכרון לשואה)	ספר C פרק 5		
7	01.05.2020-26.04.2020 (ג יום הזיכרון, ד יום העצמאות)	ספר C פרק 6	מפגש רביעי	
8	08.05.2020-03.05.2020	ספר C פרק 6		ממ״ן 12 03.05.2020
9	15.05.2020-10.05.2020 (ג ל״ג בעומר)	ספר C פרק 6,7	מפגש חמישי	

** 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	22.05.2020-17.05.2020	ספר C פרק 7		
11	29.05.2020-24.05.2020 (ו שבועות)	ספר C פרק 7	מפגש שישי	ממ"ן 22 24.05.2020
12	05.06.2020-31.05.2020	ספר C פרק 8 + פרויקט		
13	12.06.2020-07.06.2020	פרויקט וחזרה	מפגש שביעי	ממ"ן 23 07.06.2020
14	19.06.2020-14.06.2020	פרויקט וחזרה		
15	26.06.2020-21.06.2020	פרויקט וחזרה	מפגש שמיני	ממ"ן 14** 16.08.2020

מועדי בחינות הגמר יפורסמו בנפרד

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*
***** לא תינתן דחייה בהגשת הפרויקט, פרט למקרים חריגים של מילואים או מחלה ממושכת, במקרים אלו יש לתאם את מועד ההגשה עם צוות הקורס.***

2. תיאור המטלות

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

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

ממ"ן	משקל	פרקים
11	4 (ממ"ן חובה)	3,2,1
12	5 (ממ"ן חובה)	5,4
22	8 (ממ"ן רשות)	6,5,4
23	12 (ממ"ן רשות)	8,7,6
14	31 (ממ"ן חובה)	פרויקט גמר

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

הנחיות לכתיבת מטלות וניקודן

ניקוד המטלות ייעשה לפי המשקלים הבאים:

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

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

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

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

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

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

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

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

ג. תכנות - 40%

יש להקפיד על כתיבה מסודרת ומודולרית של קוד:

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

ד. יעילות התכנית והתרשמות כללית - 20%

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

ינתנו קנסות במיקרים הבאים:

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

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

3. התנאים לקבלת נקודות זכות בקורס

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

ב. ציון של לפחות 60 נקודות בבחינת הגמר.

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 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: 2020ב' **מועד אחרון להגשה:** 12.04.2020

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 (.h, .c), 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). כל תוכנית תהיה 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 תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד 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 (תוכנית ראשית בקובץ sequence. c) (50 נקודות)

עליכם לכתוב פונקציה בשם **f_sequence**, המקבלת כפרמטר מחרוזת תווים.
הפונקציה צריכה להחליט האם סדרת התווים המרכיבה את המחרוזת נתונה בסדר עולה, בסדר עולה ממש, בסדר יורד, בסדר יורד ממש או ללא סדר קבוע. רצף של תווים זהים יכול להוות סדרה "עולה"/"יורדת" - בניגוד לסדרה "עולה ממש"/"יורדת ממש". התו '\0' המסיים את המחרוזת לא נכלל בבדיקה.

על הפונקציה להדפיס את סדרת התווים, וכן להדפיס הודעה נאה המציינת האם הסדרה עולה, יורדת, וכד'. להלן מספר דוגמאות:

סדרת התווים demo היא סדרה עולה ממש.
הסדרה beef היא סדרה עולה.
הסדרה tonic היא סדרה יורדת ממש.
הסדרה spoon היא סדרה יורדת.
את הסדרה zzz והסדרה aa נגדיר כסדרות עולות.
הסדרה suddenly היא ללא סדר קבוע.

התווים במחרוזת יכולים להיות סימנים כלשהם, גם כאלה שאינם אותיות א״ב לטיניות, אך לא כולל תווים לבנים (רווח, טאב, שורה חדשה). הסדר הקובע הוא לפי קוד ה-ascii של התווים.

כמו כן, עליכם לכתוב תוכנית ראשית (הפונקציה main) שתקלוט מהמשתמש מחרוזת מהסוג המתואר לעיל, ותקרא לפונקציה **f_sequence** עם מחרוזת זו כפרמטר. הניחו שהאורך המקסימלי של המחרוזת הוא 100 תווים (לא כולל התו המסיים).

הקלט לתוכנית הוא מהמקלדת. על התוכנית להדפיס הודעת בקשה ידידותית לקלט. הניחו שהקלט תקין. אין צורך לטפל בשגיאות קלט.

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

שאלה 2 (תוכנית ראשית בקובץ match.c) (50 נקודות)

בהתאמת מחרוזות (string matching) מחפשים מופע של מחרוזת אחת (התבנית), בתוך מחרוזת אחרת (הטקסט). ייתכן ולא תהיה התאמה מושלמת בין התבנית לטקסט. בהתאמת מחרוזות ״חלשה״ מחפשים את המופע בעל מספר השגיאות (אי ההתאמות) הנמוך ביותר. כלומר מחפשים את הפוזיציה בטקסט שהחל ממנה, מספר התווים השונים בין התבנית והטקסט הוא הנמוך ביותר.

<u>לדוגמה</u>: אם התבנית היא viva והטקסט הוא vviaaa, ניסיון התאמה של התבנית החל מהפוזיציה השמאלית בטקסט (נסמנה - 0), כלומר השוואה של viva עם vvia, מורה על 2 אי-התאמות, ניסיון התאמה עם הפוזיציה 1 (viaa) מורה על אי-התאמה יחידה, ובפוזיציה 2 (iaaa) יש 3 אי-התאמות. לכן פוזיציה 1 בטקסט היא בעלת מספר אי ההתאמות הנמוך ביותר.

עליכם לכתוב פונקציה המקבלת כפרמטרים שתי מחרוזות: pattern (מחרוזת התבנית) ו- text (מחרוזת הטקסט), ומחזירה int שמציין את הפוזיציה ב-text הנותנת את מספר אי-ההתאמות הנמוך ביותר של pattern. בדוגמא לעיל הפונקציה תחזיר את הערך 1. אם יש מספר פוזיציות הנותנות את ההתאמה הטובה ביותר, יש להחזיר רק אחת מהן.

בנוסף, עליכם לכתוב תוכנית ראשית (הפונקציה main) שתקלוט מהמשתמש שתי מחרוזות (התבנית והטקסט), ותקרא לפונקציה עם מחרוזות אלה כפרמטרים. אחרי החזרה מהפונקציה, התוכנית הראשית תדפיס הודעה נאה הכוללת את מחרוזת התבנית, את מחרוזת הטקסט, ואת הפוזיציה הנותנת את ההתאמה הטובה ביותר.

הערות:

- לא ניתן להניח כי בטקסט ובתבנית אין רווחים ו/או טאבים. תווים כאלה נחשבים חלק מהמחרוזת. התו המסיים של המחרוזת לא נחשב חלק מהמחרוזת לצורך בדיקת ההתאמה.
- כדי לאפשר קלט פשוט של מחרוזות הכוללות תווים לבנים, העבירו כל מחרוזת בשורה נפרדת בקלט. השורה כולה, עד לתו "שורה חדשה" (לא כולל), תיחשב מחרוזת אחת.
- הניחו כי אורך מחרוזת הטקסט גדול-שווה מאורך מחרוזת התבנית.
- ניתן לקבוע אורך מכסימלי (סביר) של המחרוזות.

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 לקלט**. כמו כן יש **להדפיס באופן יזום מתוך התוכנית 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 התוכנית 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**.

להזכירכם: לא תינתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה ממושכת. במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

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: 2020ב' **מועד אחרון להגשה:** 03.05.2020

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 (.h, .c), 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).

כל תוכנית תהיה 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 תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד 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 adjacency.c)

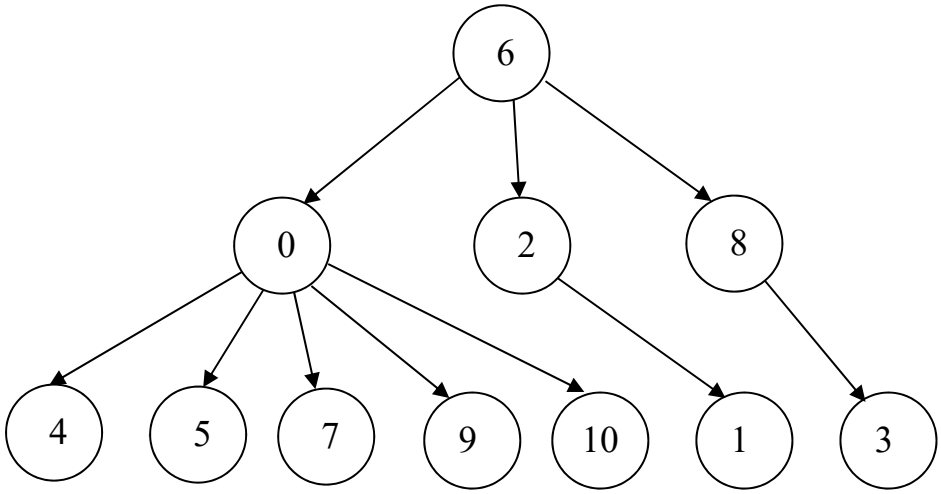
נתון עץ משרש מכוון T בעל N צמתים. "שמות" הצמתים הם האינדקסים מ-0 ועד N–1.

העץ מיוצג על ידי מטריצה A בגודל NxN באופן הבא:

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

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

בדוגמה הבאה מוצג עץ בעל N=11 צמתים.



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

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

עליכם לכתוב תוכנית לפי הדרישות שלהלן.

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

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

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

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

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

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

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

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

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

לתשומת לב:

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

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

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

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

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

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

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

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

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

להזכירכם: לא תינתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה ממושכת. במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

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: 2020ב' **מועד אחרון להגשה:** 24.05.2020

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 (.h, .c), 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). כל תוכנית תהיה 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 תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד 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 (בקבצים עיקריים mycomp.c, complex.c, complex.h)

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

<u>תזכורת מספרים מרוכבים:</u>

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

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

דוגמאות של מספרים מרוכבים:

$24.65 + (15.376)i$ $\qquad$ $87.5 - (14.3)i$ $\qquad$ $-153+24i$

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

חיבור של שני מספרים מרוכבים:

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

חיסור של שני מספרים מרוכבים:

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

כפל של מספר מרוכב במספר ממשי:

$$m * (a + bi) = ma + (mb)i$$

כפל של מספר מרוכב במספר מדומה:

$$mi * (a + bi) = mi * a + mi * bi = -mb + (ma)i$$

כפל של מספר מרוכב במספר מרוכב:

$$(a + bi) * (c + di) = a * c + a * di + bi * c + bi * di = (ac - bd) + (ad + bc)i$$

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

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

<u>משימות התוכנית:</u>

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

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

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

מפרט הפקודות המשמשות כקלטים לתוכנית:

1. הצבת מספר מרוכב במשתנה:

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

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

לדוגמה, הפקודה הבאה:

read_comp A, 5.1, 6.23

תבצע את ההצבה:

A = 5.1 + (6.23)i

הערה: זוהי הפקודה היחידה שמשנה את ערכו של משתנה מרוכב בתוכנית.

2. הדפסת ערך של משתנה מרוכב:

שם-משתנה-מרוכב print_comp

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

לדוגמה, הפקודה הבאה:

print_comp A

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

5.10 + (6.23)i

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

3. חיבור מספרים מרוכבים:

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

יתבצע חיבור של שני המספרים המרוכבים אשר במשתנים המופיעים בפקודה:

מספר-מרוכב-ב' + מספר-מרוכב-א'

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

4. חיסור מספרים מרוכבים:

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

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

מספר-מרוכב-ב' – מספר-מרוכב-א'

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

5. כפל מספר מרוכב עם מספר ממשי:

מספר-ממשי ,שם-משתנה-מרוכב mult_comp_real

יתבצע כפל של המשתנה המרוכב והמספר הממשי הנתונים בפקודה.

מספר-ממשי * מספר-מרוכב

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

6. כפל מספר מרוכב עם מספר מדומה:

מספר-מדומה ,שם-משתנה-מרוכב mult_comp_img

יתבצע כפל של המשתנה המרוכב והמספר המדומה הנתונים בפקודה.

המספר המדומה נתון בפקודה ללא הגורם i.

(i * מספר-מדומה) * מספר-מרוכב

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

7. כפל שני מספרים מרוכבים:

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

יתבצע כפל של שני המשתנים המרוכבים המופיעים בפקודה:

מספר-מרוכב-ב' * מספר-מרוכב-א'

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

8. ערך מוחלט של מספר מרוכב:

שם-משתנה-מרוכב abs_comp

יחושב ערכו המוחלט של המשתנה המרוכב שמופיע בפקודה:

| מספר-מרוכב |

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

9. סיום התוכנית:

stop

פקודה זו היא ללא פרמטרים, ומטרתה לסיים את התוכנית.

<u>המבנה התחבירי של הקלט</u>:

- כל פקודה תופיע בשלמותה בשורת קלט יחידה, כולל כל הפרמטרים. מותרות גם שורות ריקות (שורות המכילות רק תווים לבנים).
- שם הפקודה מופרד מהפרמטר הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

- בין כל שני אופרנדים יש פסיק אחד. לפני ואחרי הפסיק יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת. אסור שיהיה פסיק אחרי הפרמטר האחרון.
- יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת לפני שם הפקודה, וגם בסוף השורה (אחרי הפרמטר האחרון).
- אסור שיהיו תווי זבל בסוף השורה (למעט תווים לבנים).
- שמות הפקודות יופיעו באותיות קטנות בלבד, ושמות המשתנים באותיות גדולות בלבד.

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

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

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

להלן דוגמאות של קלט שגוי:

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

1. לפקודה:
   read_comp G, 3.1, 6.5
   יש להגיב בהודעה כגון:
   Undefined complex variable
2. לפקודה:
   read_comp a, 3.6, 5.1
   יש להגיב בהודעה כגון:
   Undefined complex variable
3. לפקודה:
   do_it A, B
   יש להגיב בהודעה כגון:
   Undefined command name
4. לפקודה:
   Add_Comp A, C
   יש להגיב בהודעה כגון:
   Undefined command name
5. לפקודה:
   read_comp A, 3.5, xyz
   יש להגיב בהודעה כגון:
   Invalid parameter – not a number
6. לפקודה:
   read_comp A, 3.5
   יש להגיב בהודעה כגון:
   Missing parameter

7. לפקודה:

read_comp A, 3.5, -3,

יש להגיב בהודעה כגון:

Extraneous text after end of command

8. לפקודה:

add_comp B

יש להגיב בהודעה כגון:

Missing parameter

9. לפקודה:

print_comp C, D

יש להגיב בהודעה כגון:

Extraneous text after end of command

10. לפקודה:

sub_comp F, , D

יש להגיב בהודעה כגון:

Multiple consecutive commas

11. לפקודה:

mult_comp_comp F D

יש להגיב בהודעה כגון:

Missing comma

12. לפקודה:

mult_comp_real, A, 2.5

יש להגיב בהודעה כגון:

Illegal comma

13. לפקודה:

mult_comp_img A, B

יש להגיב בהודעה כגון:

Invalid parameter – not a number

14. לפקודה:

abs_comp

יש להגיב בהודעה כגון:

Missing parameter

15. לפקודה:

abs_comp 2.5

יש להגיב בהודעה כגון:

Undefined complex variable

16. לפקודה:

stop A

יש להגיב בהודעה כגון:

Extraneous text after end of command

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

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

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

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

<u>דרישות נוספות:</u>

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

<u>להזכירכם</u>: לא תנתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה, במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 23

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 6,7,8

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

סמסטר: 2020ב' **מועד אחרון להגשה:** 07.06.2020

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 (.h, .c), 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). כל תוכנית תהיה 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), ניתנות להידור והרצה, ומאפשרות בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד 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 (10 נקודות)

בכל סעיף עליכם לכתוב האם נכון, לא נכון, או לפעמים נכון. עליכם לנמק את תשובתכם. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות. (כל סעיף 5 נקודות).

א. בשפת ANSI-C כל הפונקציות הן גלובליות, כלומר כל פונקציה יכולה לקרוא תמיד לכל פונקציה אחרת.

ב. הפונקציה fsize מדפיסה תמיד את גודל הקובץ.

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

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

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

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

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

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

לדוגמה, עבור קובץ קלט המכיל:

75 56 32 5 12 54 0 99 17

קובץ הפלט יכיל:

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

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

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

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

<u>להזכירכם</u>: לא תינתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה ממושכת. במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 | משקל המטלה: 31 נקודות **(חובה)**

סמסטר: 2020ב' | מועד אחרון להגשה: 16.08.2020

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 ותיעוד 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 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 באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב.
דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, וכד'.
הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזיכרון בזמן ריצתה.
כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

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

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

לכל מודל של יע"מ (כלומר לכל אירגון של מחשב) יש שפת מכונה 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 שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב **ממעבד** (יע"מ), **אוגרים** (רגיסטרים), **וזיכרון** RAM. חלק מהזיכרון משמש כמחסנית (stack).

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

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

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

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

מבנה הוראת המכונה:

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

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

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

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

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

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

להלן טבלת הפעולות:

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

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

להלן מפרט השדות במילה הראשונה בקוד המכונה של כל הוראה.

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

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

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

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

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

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

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

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

שיטות מיעון:

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

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

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

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

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

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 21 סיביות, השוכן בסיביות 3-23 של המילה. הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1, r2 בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2.
1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר <u>ללא סימן</u> ברוחב של 21 סיביות, בסיביות 3-23 של מילת המידע. הסיביות 0-2 במילת המידע הן השדה A,R,E. במיעון ישיר, ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 3-23. אם זוהי כתובת שמייצגת שורה בקובץ המקור הנוכחי (כתובת פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זוהי כתובת שמייצגת שורה בקובץ מקור אחר של התוכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data.' או 'string.', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנחית 'extern.'. התווית מייצגת באופן סימבולי כתובת בזיכרון.	השורה הבאה מגדירה את התווית x: x: .data 23 ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x). <u>דוגמה נוספת</u>: ההוראה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). הכתובת next תקודד בסיביות 3-23 של מילת המידע הנוספת.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
2	מיעון יחסי	שיטה זו רלוונטית אך ורק להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בקודי-הפעולה הבאים בלבד: jmp, bne, jsr. לא ניתן להשתמש בשיטה זו בהוראות עם קודי-פעולה אחרים. בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת המכילה את מרחק הקפיצה, במילות זיכרון, מכתובת ההוראה הנוכחית (פקודת הקפיצה) אל כתובת ההוראה המבוקשת (ההוראה הבאה לביצוע). מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 21 סיביות, השוכן בסיביות 3-23 של מילת המידע הנוספת. מרחק זה יהיה שלילי במקרה שהקפיצה היא אל הוראה שבכתובת יותר נמוכה, וחיובי במקרה שהקפיצה היא אל הוראה שבכתובת יותר גבוהה. הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון יחסי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו & ולאחריו ובצמוד אליו מופיע שם של תווית. התווית מייצגת באופן סימבולי כתובת של הוראה בקובץ המקור הנוכחי של התוכנית. ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה. יודגש כי בשיטת מיעון יחסי לא ניתן להשתמש בתווית (כתובת) שמוגדרת בקובץ מקור אחר (כתובת חיצונית).	jmp &next בדוגמה זו, ההוראה מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). נניח כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני). מרחק הקפיצה אל ההוראה בכתובת next הוא 200-, ומרחק זה יקודד בסיביות 3-23 של מילת המידע הנוספת.
3	מיעון אוגר ישיר	האופרנד הוא אוגר. לשיטת מיעון זו אין מילת מידע נוספת. מספרו של האוגר מקודד במילה הראשונה של ההוראה, בשדה המתאים: אוגר מקור/יעד.	האופרנד הוא שם של אוגר.	clr r1 בדוגמה זו, ההוראה clr מאפסת את תוכן האוגר r1.

מפרט הוראות המכונה:

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

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

קבוצת ההוראות הראשונה:
אלו הן הוראות הדורשות שני אופרנדים.

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

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

קבוצת ההוראות השניה:

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

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

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	1	איפוס תוכן האופרנד	clr r2	האוגר r2 מקבל את הערך 0.
not	5	2	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט באוגר r2 מתהפך.
inc	5	3	הגדלת תוכן האופרנד באחד.	inc r2	תוכן האוגר r2 מוגדל ב- 1.
dec	5	4	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב- 1.
jmp	9	1	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp &Line	PC←PC+distanceTo(Line) מצביע התוכנית מקבל את המען שמחושב על ידי חיבור המרחק לתווית Line עם מען ההוראה הנוכחית, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
bne	9	2	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אזי PC ← address(Line) מצביע התוכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
jsr	9	3	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערה: חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC ← address(SUBR) מצביע התכנית יקבל את כתובת התווית SUBR, ולפיכך, ההוראה הבאה שתתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	13		הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	יודפס לפלט התו (קוד ascii) הנמצא באוגר r1

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

ההוראות השייכות לקבוצה זו הן: rts, stop.

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

מבנה תכנית בשפת אסמבלי:

תכנית בשפת אסמבלי בנויה ממשפטים (statements). 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 תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התוכנית, 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 של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

<u>משפט הנחיה:</u>

משפט הנחיה הוא בעל המבנה הבא:
בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי שיתואר בהמשך. התווית היא אופציונאלית.

לאחר מכן מופיע שם ההנחיה. לאחר שם ההנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנחיה).

שם של הנחיה מתחיל בתו '.' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

יש לשים לב: למילים בקוד המכונה הנוצרות <u>ממשפט הנחיה</u> **לא מצורף השדה A,R,E**, והערך המוגדר על ידי ההנחיה ממלא את כל 24 הסיביות של המילה.

יש ארבעה סוגים (שמות) 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 בתוכנית את ההוראה:

```
mov  XYZ, r1
```

אזי בזמן ריצת התוכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

```
lea XYZ, r1
```

תכניס לאוגר 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.

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

4. ההנחיה '.extern'

להנחיה '.extern' פרמטר 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

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

<u>משפט הוראה</u>:

משפט הוראה מורכב מהחלקים הבאים:

1. תווית <u>אופציונלית</u>.
2. שם הפעולה.
3. אופרנדים, בהתאם לסוג הפעולה (בין 0 ל-2 אופרנדים).

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

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

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

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

למשפט הוראה עם שני אופרנדים המבנה הבא:

label: opcode source-operand, target-operand

לדוגמה:

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label: opcode target-operand

לדוגמה:

HELLO: bne &XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label: opcode

לדוגמה:

END: stop

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

<u>תווית:</u>

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 <u>שונות זו מזו</u>.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של אוגר) אינן יכולות לשמש גם כשם של תווית.

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

<u>מספר:</u>

מספר חוקי מתחיל בסימן אופציונלי: '–' או '+' ולאחיו סדרה של ספרות בבסיס עשרוני. לדוגמה: 76, 5–, 123+ הם מספרים חוקיים. אין תמיכה בשפת אסמבלי 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".

תפקיד השדה A,R,E בקוד המכונה

בכל מילה בקוד המכונה של הוראה (לא של נתונים), האסמבלר מכניס מידע עבור תהליך הקישור והטעינה. זהו השדה A,R,E (שלוש הסיביות הימניות 2,1,0 בהתאמה). המידע ישמש לתיקונים בקוד בכל פעם שייטען לזיכרון לצורך הרצה. האסמבלר בונה מלכתחילה קוד שמיועד לטעינה החל מכתובת 100. התיקונים יאפשרו לטעון את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

בכל מילה של ההוראה, בדיוק אחת משלש הסיביות של השדה A,R,E מכילה 1, ושתי הסיביות האחרות מאופסות. מפרט שיטות המיעון שהוצג קודם מציין איזו סיבית תכיל 1 בכל שיטת מעון.

סיבית 'A' (קיצור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה אופרנד מידי).

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

סיבית 'E' (קיצור של External) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (External) (למשל מילה המכילה כתובת של תווית שמוגדרת בקובץ מקור אחר).

אסמבלר עם שני מעברים

כאשר מקבל האסמבלר תוכנית בשפת אסמבלי, עליו לעבור על התוכנית פעמיים. במעבר הראשון, 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.

לדוגמה: האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

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

קוד המכונה של התוכנית (הוראות ונתונים) נבנה כך שיתאים לטעינה בזיכרון **החל ממען 100 (עשרוני)**.

התרגום של תוכנית תכנית המקור שבדוגמה לקוד בינארי מוצג להלן:

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

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של ההוראות והקודים הבינאריים (opcode, funct) 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 אמור להיות משויך למען 121 (עשרוני), ושהסמל K אמור להיות משויך למען 130, 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 המשויכים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של הוראות התוכנית, בערכיהם המספריים. הביצוע של שני שלבים אלה כרוך בשתי סריקות (הנקראות "מעברים") של קובץ המקור.

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משויך ערך מספרי, שהוא מען בזיכרון.

עבור הדוגמה לעיל, טבלת הסמלים היא כדלקמן:

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

במעבר השני נעשית ההמרה של קוד המקור לקוד מכונה. בתחילת המעבר השני צריכים הערכים של 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 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 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 לאוגר, או להעתקת תוכן אוגר לאוגר 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 בתוכנית, אלא יוגדר רק בהמשך התוכנית. להלן, לדוגמה, הוראת הסתעפות למען שמוגדר על ידי התווית A שמופיעה רק בהמשך הקוד:

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

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

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מיידי, 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 התוכנית, 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

בתוכנית 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 בתוכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו היא הסתעפות לא נכונה. התוכנית 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,** אם כי **בקובץ הקלט אין חובה שתהיה הפרדה כזו.** בהמשך מתואר אלגוריתם של האסמבלר, ובו פרטים כיצד לבצע את ההפרדה.

גילוי שגיאות בתוכנית המקור

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".

האסמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי stdout. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה.

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:

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

תהליך העבודה של האסמבלר

נתאר כעת את אופן העבודה של האסמבלר. בהמשך, יוצג אלגוריתם שלדי למעבר ראשון ושני.

האסמבלר מתחזק׳ שני מערכים, שייקראו להלן תמונת ההוראות (code) ותמונת הנתונים (data). מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 24 סיביות). במערך ההוראות בונה האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הנחיה מסוג ’data.‘ ו- ’string.‘).

האסמבלר משתמש בשני מונים, שנקראים IC (מונה ההוראות - Instruction-Counter), ו- DC (מונה הנתונים - Data-Counter). מונים אלו מצביעים על המקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאמה. בכל פעם כשמתחיל האסמבלר לעבור על קובץ מקור, המונה IC מקבל ערך התחלתי 100, והמונה DC מקבל ערך התחלתי 0. הערך ההתחלתי IC=100 נקבע כדי שקוד המכונה של התוכנית יתאים לטעינה לזיכרון (לצורך ריצה) החל מכתובת 100.

בנוסף, מתחזק האסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על קובץ המקור. לטבלה זו קוראים טבלת-הסמלים (symbol-table). לכל סמל נשמרים בטבלה שם הסמל, ערכו המספרי, ומאפיינים שונים, כגון המיקום (data או code), וסוג הסמל (external או entry).

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השלד של תמונת הזיכרון (הוראות ונתונים).

האסמבלר קורא את קובץ המקור שורה אחר שורה, ופועל בהתאם לסוג השורה (הוראה, הנחיה, או שורה ריקה/הערה).

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

2. שורת הוראה:

האסמבלר מנתח את השורה ומפענח מהי ההוראה, ומהן שיטות המיעון של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המיעון נקבעות בהתאם לתחביר של כל אופרנד, כפי שהוסבר לעיל במפרט שיטות המיעון. למשל, התו # מציין מיעון מידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר ישיר, וכד׳.

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

כעת האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

- אם זה אוגר – האופרנד הוא מספר האוגר.
- אם זו תווית (מיעון ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים, במידה והוא יוגדר רק בהמשך התוכנית).
- אם זה התו # ואחריו מספר (מיעון מידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תאור שיטות המיעון לעיל)

האסמבלר מכניס למערך ההוראות, בכניסה עליה מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה, את מספרי שיטות המיעון של אופרנד המקור והיעד, ואת מספרי האוגרים של אופרנד המקור והיעד במקרה של שיטת מיעון אוגר ישיר. ה- IC מקודם ב-1.

נזכור שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיביות של שיטת המיעון של אופרנד המקור יכילו 0. בדומה, אם זוהי הוראה ללא אופרנדים (rts, stop), אזי הסיביות של שיטות המיעון של שני האופרנדים יכילו 0. כמו כן, אם שיטת המיעון אינה אוגר ישיר, הסיביות של מספר האוגר הרלוונטי יכילו 0.

אם זוהי הוראה עם אופרנדים (אחד או שניים), האסמבלר "משרייין" מקום במערך ההוראות עבור מילות-המידע הנוספות הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מיעון מיידי, האסמבלר מקודד גם את המילה הנוספת המתאימה במערך ההוראות. ואילו בשיטת מיעון ישיר או יחסי, מילת המידע הנוספת במערך ההוראות נשארת ללא קידוד בשלב זה.

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 ב-1 עבור כל מספר שהוכנס.

אם בשורה 'data.' מוגדרת גם תווית, אזי התווית מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הנתונים DC <u>שלפני</u> הכנסת המספרים למערך. המאפיין של התווית הוא data.

II. 'string.'
הטיפול ב-'string.' דומה ל- 'data.', אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו במילה נפרדת). לבסוף מוכנס למערך הנתונים הערך 0 (המציין סוף מחרוזת). המונה DC מקודם באורך המחרוזת + 1 (גם התו המסיים את המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בהנחיה string. זהה לטיפול הנעשה בהנחיה 'data.'.

III. 'entry.'
זוהי הנחיה לאסמבלר לאפיין את התווית הנתונה כאופרנד כ-entry בטבלת הסמלים. בעת הפקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entries.
<u>לתשומת לב</u>: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית entry. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

IV. 'extern'.
זוהי הצהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסמבלר.
לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית extern. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

לתשומת לב: באופרנד של הוראה או של הנחית entry., מותר להשתמש בסמל אשר יוגדר 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 (100) + IC (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, תמונת הנתונים מופרדת מתמונת הוראות, וכל הנתונים נדרשים להופיע בקוד המכונה אחרי כל ההוראות. סמל מסוג data הוא תווית בתמונת הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של תמונת ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

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

במעבר השני, האסמבלר משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. במודל המכונה שלנו אלו הן מילות-מידע נוספות של פקודות, אשר מקודדות אופרנד בשיטת מיעון ישיר או יחסי. האופרנד מכיל סמל שמוגדר כפנימי או כחיצוני, ולכן בשדה ה- A,R,E הסיבית R או הסיבית E, בהתאמה, תהיה 1 (ראו גם מפרט שיטות המיעון לעיל).

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.

כאמור, אנו מחלקים את תמונת קוד המכונה לשני חלקים: תמונת ההוראות (code), ותמונת הנתונים (data). לכל חלק נתחזק מונה נפרד: IC (מונה ההוראות) ו-DC (מונה הנתונים).

נבנה את קוד המכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100.

בכל מעבר מתחילים לקרוא את קובץ המקור מההתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 100.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 17.
3. האם השדה הראשון בשורה הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data. או string. ? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין data. ערך הסמל יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בתמונת הנתונים, והגדל את מונה הנתונים DC על ידי הוספת האורך הכולל של הנתונים שהוגדרו בשורה הנוכחית. חזור ל-2.
8. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-11.
9. אם זוהי הנחית entry. חזור ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הנחית extern., הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין external. חזור ל-2.

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

מעבר שני

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

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו קודם, ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני.

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

נבצע מעבר ראשון על הקוד לעיל, ונבנה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תמונת הנתונים, ושל המילה הראשונה של כל הוראה. כמו כן, נקודד מילות-מידע נוספות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילות-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב "?" בדוגמה להלן.

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

טבלת הסמלים אחרי מעבר ראשון היא:

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

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

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

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

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטעינה.
כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

קבצי קלט ופלט של האסמבלר

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

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

- קובץ object, המכיל את קוד המכונה.
- קובץ externals, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל חיצוני (סמל שהוגדר באמצעות ההנחיה extern., ומאופיין בטבלת הסמלים כ- external).
- קובץ entries, ובו פרטים על כל סמל שמוצהר כנקודת כניסה (סמל שהופיע כאופרנד של הנחיית entry., ומאופיין בטבלת הסמלים כ- entry).

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

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות x.as , y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית 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 "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.

נציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

פורמט קובץ ה- object

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

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

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

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

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

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ- entry. בשורה מופיע שם הסמל, ולאחריו ערכו כפי שנקבע בטבלת הסמלים (בבסיס עשרוני בשבע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

פורמט קובץ ה- externals

קובץ ה-externals בנוי אף הוא משורות טקסט, שורה לכל כתובת בקוד המכונה בה יש מילת מידע המתייחסת לסמל שמאופיין כ- external. כזכור, רשימה של מילות-מידע אלה נבנתה במעבר השני (צעד 6).

כל שורה בקובץ ה-externals מכילה את שם הסמל החיצוני, ולאחריו הכתובת של מילת-המידע (בבסיס עשרוני בשבע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. 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** הנתון להלן.

```
; file ps.as

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

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

להלן הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, כפי שנבנה במעבר הראשון והשני.

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

טבלת הסמלים הסופית בגמר המעבר השני היא:

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

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ **ps.ob**:

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

הקובץ **ps.ext**:

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

הקובץ **ps.ent**:

```
MAIN 0000100
LIST 0000130
```

<u>סיכום והנחיות כלליות</u>

- גודל תוכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה <u>בלבד</u>. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).

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

- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ״ן, ללא שינויים כלשהם. כלומר, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו 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 **<u>לגלות ולדווח על כל השורות השגויות</u>** בקלט. <u>אין לעצור</u> 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 או מחלה ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

ב ה צ ל ח ה !