שפת C – תרגיל 2

מערכים סטטיים (חד ודו ממדיים), const, structs, וקריאה וכתיבה מקבצים, Makefile

<u>תאריך הגשה של התרגיל והבוחן:</u> **יום <u>ראשון</u> 13.08.17 עד שעה 23:55** [1]

1. הנחיות חשובות:

1. בכל התרגילים יש לעמוד בהנחיות הגשת התרגילים וסגנון כתיבת הקוד. שני המסמכים נמצאים באתר הקורס – הניקוד יכלול גם עמידה בדרישות אלו.
2. בכל התרגילים עליכם לכתוב קוד ברור. בכל מקרה בו הקוד שלכם אינו ברור מספיק עליכם להוסיף הערות הסבר בגוף הקוד. יש להקפיד על תיעוד (documentation) הקוד ובפרט תיעוד של כל פונקציה.
3. במידה ואתם משתמשים בעיצוב מיוחד או משהו לא שגרתי, עליכם להוסיף הערות בקוד המסבירות את העיצוב שלכם ומדוע בחרתם בו.
4. עבור כל פונקציה בה אתם משתמשים, עליכם לוודא שאתם מבינים היטב מה הפונקציה עושה גם במקרי קצה (התייחסו לכך בתיעוד). ובפרט עליכם לוודא שהפונקציה הצליחה.
5. בכל התרגילים במידה ויש לכם הארכה, או שאתם מגישים באיחור. **<u>חל איסור להגיש קובץ כלשהוא בלינק הרגיל (גם אם לינק overdue טרם נפתח)</u>**. **מי שיגיש קבצים בשני הלינקים מסתכן בהורדת ציון משמעותית.**
6. אין להגיש קבצים נוספים על אלו שתדרשו .ובפרט אין להגיש קובץ README אלא אם צוין במפורש שיש צורך בכך (לדוגמא, בתרגיל זה אין צורך להגיש).
7. עליכם לקמפל עם הדגלים Wall -Wextra -Wvla -std=c99 ולוודא שהתוכנית מתקמפלת ללא אזהרות, **תכנית שמתקמפלת עם אזהרות תגרור הורדה משמעותית בציון התרגיל.** למשל, בכדי ליצור תוכנית מקובץ מקור בשם ex1.c יש להריץ את הפקודה:

```
gcc -Wextra -Wall -Wvla -std=c99 ex1.c -o ex1
```

8. עליכם לוודא שהתרגילים שלכם תקינים ועומדים בכל דרישות הקימפול והריצה במחשבי בית הספר מבוססי מעבדי bit-64 (מחשבי האקוריום, לוי, השרת river). **<u>חובה להריץ את התרגיל במחשבי בית הספר לפני ההגשה</u>**. (ניתן לוודא שהמחשב עליו אתם עובדים הנו בתצורת bit-64 באמצעות הפקודה "uname -a" ווידוא כי הארכיטקטורה היא 64, למשל אם כתוב x86_64)
9. לאחר ההגשה, בדקו את הפלט המתקבל בקובץ ה-PDF שנוצר מהpresubmission script בזמן ההגשה. באם ישנן שגיאות, תקנו אותן על מנת שלא לאבד נקודות.
   <u>שימו לב ! תרגיל שלא יעבור את ה presubmission script ציונו ירד משמעותית</u> (הציון יתחיל מ-50, ויוכל לרדת) **<u>ולא יהיה ניתן לערער על כך.</u>**
10. בדיקת הקוד לפני ההגשה, גם על ידי קריאתו וגם על ידי כתיבת בדיקות אוטומטיות (tests) עבורו היא אחריותכם. יבדקו גם מקרי קצה.
    במידה וסיפקנו לכם קבצי בדיקה לדוגמא, השימוש בהם יהיה על אחריותכם. **במהלך הבדיקה הקוד שלכם ייבדק מול קלטים נוספים לשם מתן הציון.**
11. **הגשה מתוקנת -** לאחר מועד הגשת התרגיל ירוצו הבדיקות האוטומטיות ותקבלו פירוט על הטסטים בהם נפלתם. לשם שיפור הציון יהיה ניתן להגיש שוב את התרגיל לאחר תיקוני קוד ולקבל בחזרה חלק מהנקודות - <u>פרטים מלאים יפורסמו בפורום ואתר הקורס</u>.

---

[1] ניתן להגיש באיחור של עד יום עם קנס של 10 נקודת.

2. הנחיות ספציפיות לתרגיל זה:

1. בשאלה של MyString אין להשתמש בספריות כלשהן מלבד <stdio.h>. בחלק השני של התרגיל אתם רשאים להשתמש בכל הספריות הסטנדרטיות של C.
2. חל איסור להשתמש במערכים בגודל דינמי (VLA). שימוש כזה יוביל לפסילת הסעיף הרלוונטי.
3. עליכם לוודא שהקוד שלכם רץ באופן תקין וללא דליפות זכרון. לשם כך עליכם להשתמש בתוכנת valgrind (ראו פירוט בהמשך).
4. עליכם להשתמש בהוראות asserts לשם debugging. השימוש ב-assert נועד לבדוק את הפונקציות הפנימיות שלכם.

במקרה של שגיאה של המשתמש (העברת פרמטרים שגויה לפי ה-user interface) – צריך לדווח ל-user על השגיאה ולא להשתמש ב-assert. במקרה של העברת פרמטרים שגויה לאחת הפונקציות הפנימיות שלכם (כנראה שיש לכם באג) משתמשים ב-assert. למשל: פונקציה פנימית המקבלות מצביעים כאחד הפרמטרים שלה (אפשר לוודא כי הפרמטר אינו שווה ל-NULL), פונקציה פנימית המקבלת אינדקסים של תא(אפשר לוודא שהאינדקסים אינם שליליים), גישה למערך שגודלו ידוע(אפשר לוודא שהמשתנה חיובי ושאין חריגה מגבולות המערך) ועוד.

5. שימו לב שהאלגוריתמים שלכם צריכים להיות יעילים.
6. אתם רשאים (ולעתים אף נדרשים) להגדיר פונקציות נוספות לשימושכם הפנימי.

3. MyString (30 נקודות):

1. לרשותכם קובץ כותר (header-file) בשם MyString.h. עליכם לכתוב את הקובץ MyString.c ובו לממש את הפונקציה countSubStr המוגדרת ב header-file.
2. הפונקציה countSubStr מחזירה את מספר הפעמים שתת מחרוזת str2 מופיעה בתוך המחרוזת str1. וצורת העבודה שלה תלויה בפרמטר isCyclic:
   - אם הפרמטר הינו 0 הפונקציה תבצע חיפוש תת מחרוזת עד סוף המחרוזת str1.
   - אם הפרמטר הינו =! 0 הפונקציה תבצע את חיפוש תת המחרוזת בצורה ציקלית, דהיינו לא תעצור כאשר הגענו לסוף str1 אלא תמשיך בצורה ציקלית לתחילת המחרוזת. (בתצורת עבודה זו ייתכן כי המחרוזת str2 יותר ארוכה מהמחרוזת str1, לדוגמא: "str1 ="abcdefgh", str2 = “abcdefgha")
3. הנחיות:
   - <u>שימו לב:</u> בשאלה זאת <u>אסור לכם להשתמש בפונקציות מהספריה string.h או מכל ספריה סטנדרטית אחרת</u> (למעט (<stdio.h>). שימוש בפונקציות אלו יוביל לפסילת השאלה.
   - חיפוש מחרוזת ריקה, או במחרוזת ריקה (מחרוזת ללא תווים) יחזיר 0.
   - החיפוש הוא case sensitive, דהיינו אנו מבדילים בין אותיות קטנות לגדולות.
   - אנו סופרים רק מופעים שלמים של str2 בתוך str1 ולא חלקי מופעים.
   - אם המחרוזת שונה מ-NULL, ניתן להניח שהיא מסתיימת ב-'\0'.
   - הנקודות שעליכם להקפיד עליהן בכתיבת הפונקציה: יעילות זמן ריצה וזיכרון[2]. קוד מובנה וברור.
   - אתם רשאים להוסיף לקובץ MyString.c מימוש של פונקציית ה main שתכלול הדגמה של קריאה לפונקציה הממומשת.
   - <u>קובץ להגשה:</u> בתרגיל זה עליכם להגיש את הקובץ MyString.c המכיל את התכנה המבוקשת.

---

[2] סיבוכיות O(m*n) הינה סבירה - כאשר m ו-n הם גדלי המחרוזות.
במידה והמימוש שלכם חכם ויעיל יותר, ציינו זאת בהערות והסבירו בקצרה - אולי תזכו בנקודות בונוס :)

4. בעיית סיווג - LineSeparator (70 נקודות):

1. הבעיה של סיווג עצמים היא אחת הבעיות הידועות בתחום. דוגמא טובה להמחשת הבעיה היא הדוגמא של זיהוי תפוזים רקובים:

- חקלאים מעבירים לחברת משקאות מסוימת אלפי תפוזים לצורך הפיכתם למיץ. לצערנו הרב, חלק מהתפוזים שמגיעים אל פס היצור אינם ראויים לשימוש. החברה מחזיקה אנשים שכל תפקידם הוא לעקוב אחרי פס היצור, לזהות תפוזים שאינם ראויים לשימוש ולהוציא אותם מפס היצור באופן ידני.
- בעל החברה מחליט שהגיע הזמן להתקדם לעידן הטכנולוגי. הוא מעוניין להחליף את האנשים הנ"ל ברובוט יחיד שיזהה תפוזים שאינם ראויים לשימוש ויוציא אותם באופן אוטומטי מפס היצור. הוא פונה אליכם כדי שתפתחו עבורו רובוט מתאים.
- הבעיה: בהינתן תפוז, הרובוט צריך לדעת האם מדובר בתפוז רקוב או בתפוז ראוי לשימוש.
- פתרון ע"י למידה: נראה לרובוט סדרה של תפוזים (נניח שהוא יכול לקבל קלט של תמונה שלהם, משקל שלהם וכד'). נעביר לרובוט מידע בנוגע לכל תפוז: האם מדובר בתפוז רקוב או לא.
- הרובוט ייעזר באלגוריתם למידה כלשהו. התוצר של האלגוריתם יהיה פונקציה אשר מקבלת תפוז ומסווגת אותו לתפוז טוב או רקוב. את הפונקציה הזו יוכל הרובוט להפעיל על תפוזים חדשים שמגיעים.
- כלומר, הרובוט מקבל מספר דוגמאות של תפוזים והתיוג הנכון שלהם (רקובים / טובים), ע"פ הדוגמאות הנ"ל הוא מוצא פונקציה (בתקווה טובה) המזהה עבור כל תפוז, האם הוא רקוב או טוב.

2. באופן כללי, בעיות סיווג במערכות לומדות מאופיינות ע"י:

- קלט אימון: קלט הכולל ווקטורים של קורדינטות $\{\overline{x_i}\}_{i=1}^{m}$ והתיוג המתאים להם $\{y_i\}_{i=1}^{m}$, $y_i \in \{-1,1\}$.

עבור הדוגמא של התפוזים, כל קורדינטה ב-$\overline{x_i}$ מייצגת איזשהו מאפיין שניתן למדוד בצורה נומרית (למשל, משקל התפוז, צבע התפוז, צורתו וכד'). y_i יהיה (1-) אם התפוז ה-i הוא רקוב או (1+) אם התפוז ה-i הוא תפוז טוב.

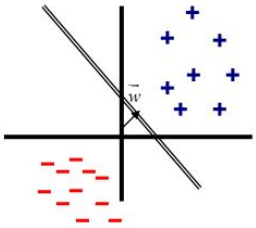
- נקרא את קלט האימון ונייצר היפותזה – פונקציה (בדרך כלל עקבית עם קלט האימון ככל האפשר) המקבלת n קורדינטות $\{x_1, x_2, ..., x_n\}$ ומחזירה סיווג של הוקטור הזה.
- לאחר האימון: נקבל דוגמאות $\{\overline{x_i}\}_{i=1}^{k}$ (במקרה שלנו, דוגמאות של תפוזים) ונחזיר סיווג מתאים ע"פ ההיפותזה שיצרנו.

3. <u>הפרדה ליניארית</u>

אנו נראה את אלגוריתם הלמידה הבסיסי והפשוט ביותר לבעיות סיווג – אלגוריתם ההפרדה הליניארית.
קיימים אלגוריתמים רבים אחרים לצורך פתרון בעיות סיווג.
בקורס iml תראו כיצד ניתן ליצור אלגוריתמי למידה חזקים מאוד מאלגוריתם ההפרדה הליניארית (כגון SVM).

4. <u>הרעיון</u>

- נתבונן על כל נקודה $\overline{x_i}$ בקלט כעל נקודה במרחב n מימדי (אם n=2 אנו מתבוננים על מישור, אם n=3 אנו מתבוננים על המרחב התלת מימדי וכו'). למען הפשטות להלן דוגמא של מרחב דו מימדי:



  - הנקודות החיוביות (מסומנות ב-'+') מייצגות קורדינטות של דוגמאות שסווגו ע"י '1+' ("תפוזים טובים" בדוגמא שלנו) והנקודות השליליות (מסומנות ב-'-') מייצגות דוגמאות שליליות ("תפוזים רקובים").
  - ההיפותזה שלנו היא 'על-מישור' ממימד n-1 (קו ישר במקרה הדו מימדי שלנו, ומישור במקרה התלת ממדי). העל מישור חוצה את המרחב ל-2 – כל נקודה שתצא מצד אחד של העל-מישור נתייג ב- (1+), כל נקודה שתצא מהצד השני של העל מישור נתייג ב- (1-).
- מהדוגמא קל להבחין בחיסרון המשמעותי ביותר של האלגוריתם שלנו: למרבית הקלטים במציאות לא קיימת הפרדה ליניארית. כאמור, יש דרכים לשדרג את אלגוריתם ההפרדה הליניארית ולקבל אלגוריתמים חזקים ממנו.
- אנו נניח כי ה-'על מישור' המפריד עובר דרך הראשית (קל להרחיב את האלגוריתם כך שהוא ילמד 'על מישור' מפריד שעובר במקום כלשהו).

5. <u>האלגוריתם</u>

1. תחילה נשים לב כי ניתן לאפיין כל על מישור מפריד (בכל מרחב) ע"י הווקטור המאונך לו ($\overline{w}$ בדוגמא למעלה).
2. יהי $\overline{w}$ ווקטור אנכי לעל מישור מפריד. תהי $\overline{x}$ נקודה כלשהי במרחב.
   - אם $\langle \overline{x}, \overline{w} \rangle > 0$ אזי הנקודה נמצאת בצד החיובי של העל מישור המפריד [תתויג ב-(1+)]
   - ואם $\langle \overline{x}, \overline{w} \rangle \le 0$ הנקודה נמצאת בצד השלילי של העל מישור [ותתויג ב-(1-)].
3. הביטוי $\langle \overline{x}, \overline{w} \rangle$ מייצג לכל אורך השאלה את המכפלה בפנימית של הווקטורים

$$\overline{x} \in R^n,\ \overline{w} \in R^n\ : \langle \overline{x}, \overline{w} \rangle = \sum_{i=1}^{n} x_i w_i$$

4. נסמן ב-m את מספר הנקודות בקלט האימון $\{(\vec{X}_j, y_j)\}_{j=1}^{m}$. האלגוריתם למציאת $\overline{w}$ עובד באיטרציות.
   - אתחול: $\overline{w} = \overline{0}$
   - איטרציות $j = 1, 2, ..., m$:
     - חשב את $y'_j = \langle \overline{x}, \overline{w} \rangle >? 0$
     - אם $y_j == y'_j$ לא נעשה דבר. אחרת $\overline{w} = \overline{w} + y_j \overline{x_j}$
   - <u>החזר</u> $\overline{w}$

6. <u>המשימה</u>

1. עליכם לכתוב תוכנית בקבצים LineSeparator.c ו- LineSeparator.h.
2. התוכנית מקבלת כפרמטר שם של קובץ המכיל את דוגמאות האימון ונתונים לתיוג. התוכנית תמצא וקטור הפרדה $\overline{w}$ (בהתאם לדוגמאות האימון) ותדפיס את התיוג המתאים של כל שאר הנתונים לתיוג.
3. מבנה הקובץ:
   - שורה ראשונה – המימד בו אנו עובדים (n).
   - שורה שניה – מספר הדוגמאות בקלט האימון (m).
   - m שורות הבאות – תיאור של נקודה במרחב ה-n מימדי והתיוג המתאים לה.
     - כל נקודה מתוארת ע"י הפורמט: $x_1, x_2, ..., x_n, y$
     - למשל עבור מרחב תלת מימדי (n=3), השורה: 3,3.5,9,1
       מתארת שהנקודה (3,3.5,9) מתויגת ע"י (+1).
     - שימו לב, y יכול לקבל רק (+1) או (-1).
   - בהמשך הקובץ מופיעים מספר לא ידוע מראש של דוגמאות שברצוננו לתייג.
     - כל דוגמא ניתנת ע"י: $x_1, x_2, ..., x_n$
     - ונרצה לתייגה ב-(+1) או ב-(-1) בהתאם ל$\overline{w}$ שקיבלנו ע"י הרצת אלגוריתם הלמידה על קלט האימון.
       כלומר, נדפיס את: $\langle \overline{x}, \overline{w} \rangle >? 0$.
4. <u>הנחות מפשטות:</u>
   - כל המספרים בקלט מופרדים ע"י פסיק יחיד (ללא רווחים / טאבים / מפרידים אחרים).
   - כל שורה (כולל השורה האחרונה) מסתיימת ב-EOL.
   - אורך שורות הקלט הוא לכל היותר 400 (שימו לב כי ההנחה האחרונה גוררת חסם על מימד המרחב).
   - קובץ הקלט חוקי: התיוגים אכן מקבלים (+1) או (-1), הנקודות הם מספרים ממשיים חוקיים (מטיפוס double), המימד של כל הנקודות זהה ותואם את המימד שכתוב בתחילת הקובץ.
   - n (מימד המרחב) ו-m (מספר דוגמאות האימון) הם מספרים מטיפוס int
   - n>1 ; m>=1
5. <u>הנחיות ורמזים:</u>
   - האלגוריתם למציאת וקטור הפרדה מפורט בסעיף 4.5.
   - אתם נדרשים להשתמש ב-struct לצורך תיאור נקודה בקלט האימון (מערך + תיוג מתאים).
   - וודאו שאתם מעבירים את ה-struct באופן יעיל (לא מעתיקים אותו).
   - עליכם להוסיף ל HeaderFile LineSeparator.h את כל הקבועים המשמעותיים בהם אתם משתמשים (למשל הקבוע המתואר בסעיף הבא).
   - כל נקודה במרחב מאופיינת ע"י מערך ב-struct.

- בתרגיל זה, נקודה הנמצאת על המישור המפריד מתויגת באופן שלילי (1-). לשם כך, עליכם להשתמש ברמת דיוק של 5 ספרות אחרי הנקודה (0.00001). כלומר ערך בתווך שבין 0.00001 ל-0.00001- (לא כולל הקצוות) ייחשב ל-0 ("המצאות על המישור").
- בצעו מעבר יחיד על קובץ הקלט לצורך קריאת הנתונים.
- אל תשמרו נתונים בזיכרון ללא לצורך (לדוגמא, אין צורך לשמור את קלט האימון בזיכרון).
- מוטל עליכם לבדוק שהתוכנית מורצת עם פרמטר שם קובץ הקלט, ושפתיחת הקובץ אכן הצליחה (הקובץ קיים).
- השתמשו בפונקצית assert כדי להבטיח נכונות הקוד שלכם. אל תמחקו את השימוש בפונקציה כאשר אתם מסיימים את שלב ה-debugging – אם ה-user לא ירצה להשתמש בפונקציה, הוא יבצע קומפילציה לקוד שלכם עם הדגל –DNDEBUG.
- הימנעו משכפול קוד, השתדלו להשתמש בפונקציות ספריה מוכנות של המערכת ולהימנע מלהמציא את הגלגל מחדש.
- רמז לביצוע ה-parsing: ניתן לקרוא את השורות לתוך מחרוזת (למשל ע"י fgets), לאחר מכן לחפש הופעות ',' בשורה ולהשתמש בהתאם בפונקציה sscanf על המחרוזת.

  פונקציות שיכולות לסייע: strncpy, strpbrk, strtok.

  אנחנו נותנים לכם חופש מוחלט בבחירה כיצד לבצע parsing –יש הרבה מאוד דרכים לביצוע הקריאה מהקובץ שכולם יתקבלו (כל עוד הם עושים את העבודה ללא תקלות).
- אינכם רשאים להשתמש בפקודה scanf שהיא איננה פונקציה בטוחה ואינכם יכולים להשתמש בפונקציות atol, atoi, atof,atofl ודומותיהן משום שלא תמיד ניתן לדעת אם הפונקציה הצליחה. שימו לב שאתם מכירים כל פונקציה בה אתם משתמשים ושאתם בודקים עבור כל פונקציה שהיא הצליחה.

  הפונקציה feof מחזירה ערך ששונה מאפס אחרי שנכשלתם בביצוע קריאה מהקלט בשל הגעה לסוף הקובץ. היא לא יודעת לבדוק בעצמה אם התו הבא בקלט הוא סימן סוף הקובץ. מומלץ לא להתבסס עליה ולבדוק ישירות את הערך המוחזר מ-sscanf / fgets.
- אין להשתמש במשתנים סטאטיים / גלובליים.
- להזכירכם, בעת קריאה לפונקציות עליכם לוודא שהן רצו באופן תקין ללא שגיאות.

4. עבודה עם valgrind:

1. ניהול זיכרון ב-C הוא נושא רגיש ומועד לפורענות – יש הרבה אפשרויות לטעות (לא להקצות מספיק זיכרון, לשכוח לשחרר זיכרון, להשתמש במצביעים שמצביעים לזבל וכו'). כמובן שהקומפיילר לא ידווח על שגיאה בכל המקרים הללו. יתכן שתגלו את השגיאות הללו בזמן ריצה, אך יתכן גם כי התוכנה תעבוד אצלכם "במקרה" והבעיות יתגלו דווקא בביתו של הלקוח.
2. ישנו מבחר די גדול של תוכנות בשוק שמטרתם לסייע באיתור בעיות זיכרון בקוד לפני שחרורו אל הלקוח. אנו נשתמש בתוכנת valgrind, שיחסית לתוכנה חינמית, נותנת תוצאות מעולות. אתם מתבקשים להריץ את valgrind עם התוכנה שלכם, ולהגיש את הפלט שלה בקובץ בשם valdbg.out.
3. כדי להריץ את valgrind עליכם לבצע קומפילציה ו-linkage לקוד שלכם עם הדגל '-g' (הן בשורת הקומפילציה והן בשורת ה-linkage). לאחר מכן הריצו valgrind:

```
> valgrind --leak-check=full --show-possibly-lost=yes
        --show-reachable=yes –undef-value-errors=yes  ProgramName
```

4. אם קיבלתם הודעת שגיאה, יתכן שתצטרכו לבצע שינוי הרשאות:

```
> chmod 777 ProgramName
```

5. כמובן שאם valgrind דיווח על בעיות עם הקוד שלכם, עליכם לתקן אותן.
6. היעזרו ב-tutorial הקצרצר של valgrind שבאתר הקורס.

5. הערות למשימות התכנות:

1. התכניות יבדקו גם על סגנון כתיבת הקוד וגם על פונקציונאליות, באמצעות קבצי קלט שונים (תרחישים שונים להרצת התכניות). הפלט של פתרונותיכם יושווה (השוואת טקסט) לפלט של פתרון בית הספר. לכן עליכם להקפיד על פורמט הדפסה מדויק, כדי למנוע שגיאות מיותרות והורדת נקודות.
2. במידה וסיפקנו לכם קבצי קלט/פלט לדוגמה (אלו מהווים רק חלק קטן מקבצי הקלט-פלט שנשתמש בהם, כתבו לעצמכם בדיקות נוספות), עליכם לוודא שהתכנית שלכם נותנת את אותו הפלט בדיוק.
3. על מנת לעשות זאת הריצו את תכניתכם עם הקלט לדוגמה (באמצעות האופרטור ">") ונתבו את הפלט של תכניתכם לתוך קובץ (באמצעות האופרטור "<") באופן הבא:

```
ProgramName < inputFile > myOutputFile
```

4. ואז השוו את קובץ הפלט שנוצר לכם עם קובץ הפלט המתאים של פתרון בית הספר, באמצעות הפקודה diff

   diff הנה תוכנה להשוואה טקסטואלית של שני טקסטים שונים. בהינתן שני קבצי טקסט להשוואה (1.txt ,2.txt) הפקודה הבאה תדפיס את השורות אשר אינן זהות בשני הקבצים:

```
diff 1.txt 2.txt
```

   במידה והקבצים זהים לחלוטין, לא יודפס דבר.
   קראו על אפשרויות נוספות של diff בעזרת הפקודה man diff.
   לחלופין אתם יכולים גם להשתמש בתוכנה tkdiff אשר מראה גם את השינויים ויזואלית.

5. כמו כן, אתם יכולים גם להשוות ישירות באופן הבא:

```
ProgramName < inputFile | diff expected.out
```

6. אם ישנם מקרים שהוראות התרגיל לא מציינות בבירור כיצד התכנית צריכה להתנהג, הביטו בקבצי הקלט וקבצי הפלט לדוגמה שניתנים לכם ובדקו אם התשובה לשאלתכם נמצאת שם. כמו כן, היעזרו בפתרון בית הספר, הריצו עליו את הטסטים שלכם והשוו להתנהגות תוכניתכם.

6. חומר עזר:

1. את פתרון הבית ספר ניתן למצוא ב:

~plabc/www/ex2/schoolSol.tar

2. את הקבצים המסופקים לצורך התרגיל ניתן למצוא ב:

~plabc/www/ex2/ex3_files.tar

3. דוגמאות לטסטים ניתן למצוא ב:

~plabc/www/ex2/tests_examples.tar

7. הגשה:

1. עליכם להגיש קובץ tar בשם ex2.tar המכיל **<u>רק</u>** את הקבצים הבאים:
   - קבצי פלט של valgrind:
     - valdbg_MyString - פלט הריצה עם valgrind של MyString עם קובץ הדרייבר MyStringExample.c שסופק לכם.
     - valdbg_LineSeparator - פלט הריצה עם valgrind של LineSeparator עם קובץ הקלט LineSeparator.in שסופק לכם.
   - קובץ Makefile התומך לפחות בפקודות הבאות:
     - make LineSeparator - יצירת קובץ ריצה בשם LineSeparator (ללא בדיקות debug[3]).
     - make LineSeparator.o - יצירת LineSeparator.o (ללא בדיקות debug).
     - make MyStringExample - יצירת קובץ ריצה בשם MyStringExample (ללא בדיקות debug).
     - make MyString.o - יצירת MyString.o (ללא בדיקות debug).
     - make - הרצת MyStringExample (ללא בדיקות debug).
     - make clean - ניקוי כל הקבצים שנוצרו באמצעות פקודות ה-Makefile (וניתן לשחזר באמצעות קריאה מחודשת לפקודות ה- make המתאימות)
   - LineSeparator.c
   - LineSeparator.h
   - MyString.c
   - extension.pdf - <u>רק</u> במקרה שההגשה היא הגשה באיחור.
2. לפני ההגשה, פתחו את הקובץ ex2.tar בתיקיה נפרדת וודאו שהקבצים מתקמפלים ללא שגיאות וללא אזהרות.
3. מומלץ מאוד גם להריץ בדיקות אוטומטיות וטסטרים שכתבתם על הקוד אותו אתם עומדים להגיש. בנוסף, אתם יכולים להריץ בעצמכם בדיקה אוטומטית עבור סגנון קידוד בעזרת הפקודה:

~plabc/www/codingStyleCheck <file or directory>

כאשר <directory or file> מוחלף בשם הקובץ אותו אתם רוצים לבדוק או תיקייה שיבדקו כל הקבצים הנמצאים בה (שימו לב שבדיקה אוטומטית זו הינה רק חלק מבדיקות ה codingStyle)

4. דאגו לבדוק לאחר ההגשה את קובץ הפלט (submission.pdf) וודאו שההגשה שלכם עוברת את ה-presubmission script ללא שגיאות או אזהרות.

~plabc/www/ex2/presubmit_ex2

בהצלחה!

---

[3] כלומר עם הדגל NDEBUG.