מעבדה מתקדמת בשפת C

תרגיל מספר 2

<u>תזכורת:</u> חובה להגיש תרגילים בזוגות, למעט מקרים שאושרו פרטנית מול סגל הקורס. התרגילים נבדקים אוטומטית בעזרת מערכת לזיהוי העתקות, כולל מול הגשות מסמסטרים קודמים.

<u>מועדי ההגשה:</u> (ראו הנחיות בסוף קובץ זה לגבי השלבים השונים)

שלב 1: 28.11, 23:59

שלב 2: 12.12, 23:59

גירסה סופית: 19.12, 23:59

<u>התרגיל:</u>

בתרגיל זה נכתוב תוכנית בשם my_grep המקבלת כקלט ביטוי ושם קובץ. אם התוכנית לא קיבלה שם קובץ, היא קוראת קלט מ-stdin. התכנית סורקת את הקובץ שורה-שורה ומדפיסה רק את השורות המכילות את הבטוי.

פרמטרים אותם התכנית יכולה לקבל (התוכנית יכולה לקבל כמה מהפרמטרים ביחד):

A NUM- התכנית תדפיס גם NUM שורות אחרי השורה המכילה את הביטוי.

b- לפני כל שורה יודפס מספר הבתים מתחילת הקובץ ועד תחילת השורה, ולאחר המספר יודפסו נקודותיים.

c- במקום להדפיס את השורות, יודפס רק מספר השורות.

i- התכנית תתעלם מההבדל בין אותיות גדולות וקטנות.

n- הדפס לפני כל שורה את מספרה בקובץ הקלט. שורות שמודפסות עקב שימוש באופציה A- יודפסו עם מקף בין מספר השורה לתוכן השורה. שורות שהיו מודפסות גם ללא האופציה הזו, יודפסו עם נקודותיים בין מספר השורה לתוכן השורה.

v- הדפס רק את השורות שלא מכילות את הביטוי.

x- הדפס רק שורות שמכילות את הביטוי, ולא מכילות דבר פרט לבטוי.

הביטוי אותו מנסה התכנית למצוא יכול להכיל אותיות גדולות וקטנות, ספרות, וכן את כל התווים הנראים הרגילים כלומר כל התווים בטבלת ASCII בתחום 33 עד 126 (דצימלי).

חפוש התוים .()[]{}|\ יוברח ע"י באקסלאש \. הופעה של אחד מהתווים הללו בביטוי (כשהם מוברחים, או כשהם לא מוברחים) מחייבת שימוש במתג E-, ומחייבת שהביטוי יהיה מוקף במרכאות.

בנוסף ניתן יהיה לעשות שמוש בבטויים רגולרים בעזרת הארגומנט –E, שחייב להופיע מיד לפני הביטוי, כאשר הביטוי מוקף במרכאות. הביטוי עצמו יכול להיות מורכב כמפורט:

(str1|str2) – משמעו חפש את הבטוי str1 או str2

. (נקודה) – כל תו יותאם. לא יכולות להופיע נקודות בתוך סוגריים.

[x-y] – מתאים לכל תו בין x ל- y כולל.

נקודות וסוגריים מרובעים ועגולים לא יכולות להופיע בתוך סוגריים מרובעים או עגולים.

דוגמאות:

הבטוי [\[(1|)[0-9]\] מתאים לכל המספרים הקטנים מעשרים (כולל אפס) המוקפים בסוגריים מרובעים.

הרצת my_grep –n –i “o\.pdf” 2013.html על הקבץ הבא:

```
<HTML>
<HEAD>
<TITLE>Yuval Shavitt (Advanced Lab in C)</TITLE>
</HEAD>
<body>
   <h1>Advanced Laboratory in C - 2013</h1>
<ul>
         <li> <a href="C-intro.pdf">Revisiting C</a>
         <li> <a href="Mudularity.docx">Modularity</a>
         <li> <a href="FileIO.pdf">I/O and files in C</a>

</ul>
```

תדפיס:

```
8:      <li> <a href="C-intro.pdf">Revisiting C</a>
10:     <li> <a href="FileIO.pdf">I/O and files in C</a>
```

לא בטוחים לגבי ההתנהגות המצופה של התוכנית במצב מסוים? הריצו grep בנובה וראו מה ההתנהגות המצופה. שאלות בסגנון "מה צריכה להיות ההתנהגות במצב הזה" לא יענו, למעט אם הסיבה להתנהגות של grep האמיתית לא ברורה לכם. במקרה כזה, ציינו זאת בבקשה בשאלה, ופרטו מה בדיוק בהתנהגות מבלבל אתכם. ניתן גם לחפש בגוגל (או להריץ בנובה) man grep כדי להגיע לתיעוד.

<u>הערות לגבי הקוד:</u>

1. אין לכתוב לקבצים במהלך התוכנית שלכם. כלומר האסטרטגיה של לטפל ב-stdin ע"י כתיבה של התוכן שלו לקובץ "ומשם כבר פתרנו", לא חוקית. האסטרטגיה הזו לא עובדת במצב שאין לכם הרשאות כתיבה.
2. אין להשתמש בפונקציות ספריה המתעסקות בביטויים רגולריים, כגון regcomp וכו'.
3. כל פעולה שאתם עושים בקוד, צריכה להתבצע במקום אחד בדיוק. בפרט, <u>אין טעם לממש שתי פונקציות שבודקות התאמה, אחת לביטוי רגולרי ואחת לביטוי פשוט, אם הפונקציה של הביטוי הרגולרי יכולה לטפל גם בביטוי פשוט</u>. מצב כזה מהווה שכפול קוד, כי הוא גורר את כל הבעיות שקורות כשמשכפלים קוד, כפי שדיברנו בכיתה. הוא גם גורר הורדת נקודות.

<u>נהלי הגשה</u>

תחת התיקיה c_lab שיצרתם לטובת הגשת התרגיל הקודם, יש ליצור תיקיה בשם ex2. יש לשים בתיקיה תת-תיקיות כמתואר מטה עם קבצי הקוד וקובץ ההרצה, ליצור בכל תת-תיקיה עותק של קובץ ה-README מהתרגיל הקודם, ולוודא שיש להם הרשאות 755. יש לשמור את התרגיל המוגש באותה תיקיה עד לסיום הקורס - ייתכן שתידרשו להשתמש בו בתרגילים שיגיעו בעתיד.

<u>פיתוח אינקרמנטלי</u>

במסגרת התרגיל, נתרגל פיתוח בשיטה אינקרמנטלית:

1. עד מועד ההגשה של שלב 1 (ראו בראש התרגיל) צריכה להיות בתיקיית התרגיל תת תיקיה בשם beta1 העוברת בהצלחה את הטסטים המורצים ע"י הפקודה:
~nimrodav/grep_tests/beta1.sh
2. עד מועד ההגשה של שלב 2 (ראו בראש התרגיל) צריכה להיות בתיקיית התרגיל תת תיקיה בשם beta2 העוברת בהצלחה את הטסטים המורצים ע"י הפקודה:
~nimrodav/grep_tests/beta2.sh

3. עד מועד ההגשה של הגירסה הסופית (ראו בראש התרגיל) תיקיית התרגיל הראשית צריכה לעבור את כל הטסטים המורצים ע"י הפקודה
~nimrodav/grep_tests/run_all.sh

יש להריץ את הפקודות בתיקיה (או תת-תיקיה) בה נמצא קובץ ההרצה בעזרת השורות הרשומות מעלה. אם הפקודה לא הדפיסה אף שורה למסך, הכל בסדר. אם היא הדפיסה שורות למסך, הריצו את כל אחת מתת-הבדיקות שהיא מריצה ובידקו מה לא תקין (עשו cat לקובץ הזה כדי לראות מה הן תת הבדיקות).
הפקודה בודקת שההרשאות של כל הקבצים בתיקיה, ושל התיקיה עצמה, הן בדיוק 755. אנא ודאו שאלה בדיוק ההרשאות, כי הפקודה תדפיס למסך כל קובץ שההרשאות שלו אינן 755. מומלץ לוודא גם ידנית במועד ההגשה שכל ההרשאות הן אכן 755.
הערה: בדיקת התרגיל, כולל תיקיות ה-beta, תיעשה במועד ההגשה הסופי שלו.
הבדיקה תוודא שתיקיות ה-beta לא עודכנו לאחר מועדי ההגשה שלהן.

ימי חסד לגירסאות הבטא
לצורך הגשת גירסאות הבטא, עומדים לרשותכם סה"כ ארבעה ימי חסד נפרדים, כלומר איחורים בגירסאות הבטא לא יורדים ממכסת ימי החסד הרגילים. איחור בהגשת הגירסה הסופית של התרגיל, כן יורד ממכסת ימי החסד הרגילים.

שימוש ב-clang-format, clang-tidy, valgrind
גירסת ההגשה של התרגיל צריכה להיות נקייה מאזהרות של valgrind ו-clang-tidy, ואחרי שהקוד עבר פירמוט בעזרת clang-format.
גירסאות הבטא גם הן צריכות להיות נקיות מאזהרות של valgrind, אך לא חייבות להיות נקיות מאזהרות של clang-tidy, ולא חייבות להיות לאחר פירמוט בעזרת clang-format.
הטסט האוטומטי של התרגיל אוכף זאת.

שימוש ב-Makefile
יש ליצור Makefile, התומך בקימפול של כל מודול בנפרד, כפי שלמדנו בכיתה.
ה-Makefile צריך גם לתמוך במטרות test, all ו-clean, שהן מסוג .PHONY.
מטרת test מריצה את קובץ הטסט של התרגיל. שימו לב: make עשויה לא להכיר את התו תילדה ~ עבור תיקיית בית. ניתן לתת נתיב מלא אל קובץ הטסט כפי שהוא יושב בתיקיה שלי, או להעתיק אותו אליכם ולהריץ אותו מתוך התיקיה שלכם.

קובץ DESIGN

בגירסת ההגשה הסופית, יש ליצור בתיקיה קובץ בשם DESIGN, המכיל שורה עבור כל מודול. כל שורה נפתחת בשם המודול, וכוללת משפט שמסביר מה תוכן המודול. שימו לב: אם מילות המפתח בהסבר לא מופיעות בשם המודול, זה סימן שאולי שם המודול לא מתאים.

מעבר נוסף על המסמך "סדר פעולות בדיקת קובץ H" ועל ההוראות

אחרי שסיימתם לכתוב את הקוד ולפני הגשת התרגיל, אנא קיראו בשנית את המסמך "סדר פעולות בדיקת קובץ H" שמופיע במודל, ו-ודאו שקבצי ה-H שלכם עומדים בקריטריונים שהוא מגדיר.

לנוחותכם, ועל מנת להקל על שיתוף הפעולה בין שותפים, אנא צרו קובץ בשם SDP בתיקיית ההגשה, אשר מכיל שורה אחת ובה רצף האותיות SDP. כך תוכלו להיות בטוחים שלפחות אחד מהשותפים עבר על הקובץ הנ"ל, ושלא תאבדו נקודות סתם. הטסט האוטומטי של התרגיל אוכף זאת.

כמו כן, יש ליצור בתיקיה קובץ בשם UNIFORM_MATCHING, על מנת להבהיר שקראתם את הערה מס' 3 הנ"ל, לגבי זה שאין טעם לממש שתי פונקציות התאמה.

כמו כן, יש ליצור קובץ נוסף בתיקיה בשם COPYING_WILL_BE_REPORTED, שמכיל שורה אחת ובה שם הקובץ, COPYING_WILL_BE_REPORTED. זאת על מנת להבהיר לי וגם לכם שאתם מבינים שמקרים של העתקה ידווחו לועדת משמעת.

בונוס - קבצי טסט

אם אתם מעוניינים לקבל בונוס: בגירסת ההגשה הסופית, עבור כל מודול מלבד המודול שמכיל את הפונקציה main, יש לצרף קובץ C הבודק אותו. מומלץ להיעזר במצגת "The Bowling Game Kata in C" שכיסינו בכיתה. שם קובץ הטסט חייב להיות שם המודול ואחריו

_test.c

For example: If the module is named frob_gizmo.c, the test file should be named frob_gizmo_test.c

קובץ הבדיקה מכיל פונקציית main, המריצה בדיקות (במקרה שיש יותר מבדיקה אחת, אפשר לבצע כל בדיקה בפונקציה נפרדת, ולקרוא לכל אחת מהפונקציות הללו מ-main).

עבור כל קובץ בדיקה, יש ליצור build target ב-Makefile, ששמה כשם קובץ הבדיקה ללא הסיומת .c. הקובץ הנוצר הוא קובץ הרצה, שמריץ את הבדיקות.

במסגרת הבדיקות, כל פונקציה חיצונית של המודול צריכה להיקרא לפחות פעם אחת.

ביצוע מלא של מטלת הבונוס יעניק 15 נקודות נוספות לציון.

בהצלחה!