234124 - מבוא לתכנות מערכות
תרגיל בית מספר 1
סמסטר אביב 2022

תאריך פרסום: 10.04.2022
תאריך הגשה: 1.5.2022 בשעה 23:59
מתרגלים אחראים לתרגיל: אסף במברגר, אחלאם אבוגוש.

```
  /^ ^\
 / 0 0 \
V\ Y /V
 / - \
 |     \
 || (__V
```

1 הערות כלליות

- **תרגיל זה מהווה 6% מהציון הסופי**
- התרגיל להגשה **בזוגות** בלבד.
- מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה או בסדנות. לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.
- שימו לב: לא תינתנה דחיות במועד הגשת התרגיל פרט למקרים חריגים. **תכננו את הזמן בהתאם**.
- ראו את התרגיל עד סופו לפני שאתן מתחילות לממש. חובה להתעדכן בעמוד ה-F.A.Q של התרגיל, הכתוב שם מחייב.
- העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם זאת – מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.
- קבצי התרגיל נמצאים בGitHub repository הבא: https://github.com/CS234124/ex1

2 חלק יבש – זיהוי שגיאות בקוד

2.1 *סעיף א'*

מצאו 6 שגיאות תכנות ו-2 שגיאות קונבנציה[1] (code convention) בפונקציה הבאה. מטרת הפונקציה היא לשכפל מספר פעמים את המחרוזת המתקבלת לתוך מחרוזת חדשה. למשל, הקריאה stringDuplicator("Hello",3) תחזיר את המחרוזת "HelloHelloHello". במקרה של שגיאה בריצת הפונקציה, הפונקציה תחזיר NULL. מותר להניח שהקלט תקין, כלומר אין צורך בבדיקה תקינות קלט כפי שלמדתם בתרגולים.

```
#include "stdlib.h"
#include "string.h"
#include "assert.h"

char* stringDuplicator(char* s, int times){
      assert(!s);
      assert(times > 0);
      int LEN = strlen(s);
      char* out = malloc(LEN*times);
      assert(out);
      for (int i=0; i<=times; i++){
      out = out + LEN;
      strcpy(out,s);
      }
      return out;
}
```

2.2 *סעיף ב'*

כתבו גרסה מתוקנת של הפונקציה.

[1] ראו מסמך "Code Conventions.pdf" באתר הקורס

3 חלק רטוב

בתרגיל זה נממש רשימה מקושרת שמכילה תוויASCII (מטיפוס char). ממשק הרשימה מובא בסעיף 3.1. מימוש הרשימה יהיה באמצעות ייצוג מיוחד שנקרא *קידוד אורך חזרה* - RLE (Run-length Encoding), שאותו נתאר בסעיף 3.2. בסעיפים 3.3 ו-3.4 נעזר ברשימה לצורך ייצוג מחרוזות מיוחדות שיוצרות תמונה על המסך (ASCII ART).

3.1 *ממשק רשימה מקושרת עם Run-length Encoding*

בחלק זה נתאר את הממשק של רשימת RLE. להלן הפעולות שעל הרשימה לתמוך בהן:

1. **RLEListCreate()** – יצירת רשימת RLE ריקה חדשה.
2. **RLEListDestroy(RLEList list)** – הריסת רשימת RLE המתקבלת כארגומנט תוך שחרור מסודר של כל הזיכרון שבשימוש.
3. **RLEListAppend(RLEList list, char value)** – הוספת תו בסוף הרשימה.
4. **RLEListSize(RLEList list)** – החזרת מספר התווים הכולל ברשימה.
5. **RLEListRemove(RLEList list, int index)** – מחיקת התו באינדקס הנתון מהרשימה.
6. **RLEListGet(RLEList list, int index, RLEListResult *result)** – החזרת התו הנמצא באינדקס הנתון.
7. **RLEListMap(RLEList list, MapFunction map_function)** – הפונקציה מקבלת רשימה ומצביע לפונקציית מיפוי המקבלת תו ומחזירה תו. הפונקציה משנה כל תו c ברשימה לתו המוחזר מהקריאה: $mapFunction(c)$.
8. **RLEListExportToString(RLEList list, RLEListResult* result)** – החזרת כל האיברים ברשימת RLE כמחרוזת אחת, לפי הפורמט הבא:

$$< char >< number\ of\ consecutive\ appearances >$$
$$< char >< number\ of\ consecutive\ appearances >$$

...

בפורמט זה, כל התווים ברשימה מופיעים בזה אחר זה במחרוזת (תו אחד בכל שורה), כאשר תו שחוזר על עצמו מספר פעמים <u>ברצף</u> מקבל שורה אחת בלבד במחרוזת, שבה כתוב התו, ולאחריו (ללא רווח) מספר הפעמים שבהן הוא מופיע ברצף ברשימה. לדוגמה, רשימה שמכילה את התווים WWWABBCC (משמאל לימין) תהפוך למחרוזת הבאה (כולל תווי ירידת שורה):

W3
A1
B2
C2

3.2 *מימוש רשימת RLE*

מימוש הרשימה המקושרת יהיה באמצעות קידוד שנקרא RLE, שמטרתו לחסוך בזיכרון. RLE הינה טכניקת דחיסה פשוטה למחרוזות, שבה תו שחוזר על עצמו מספר פעמים ברצף במחרוזת, מקודד על ידי ציון התו ומספר החזרות שלו בלבד. לדוגמה, עבור המחרוזת הבאה:

WWWWWWBBBBBBWWWWWWWWWBBBWWWWWWWWWWWW

המידע הדחוס ייראה כך:

W6, B6, W9, B3, W12

בתרגיל זה נממש רשימה מקושרת שמקודדת באמצעות RLE, כלומר, שבה כל node מייצג תו + מספר הפעמים שהוא מופיע ברצף במחרוזת. כפי שניתן לראות בדוגמה, במקום רשימה מקושרת עם 36 nodes (אחד לכל תו) בייצוג הרגיל, ברשימת RLE נצטרך רק 5 nodes על מנת לייצג את אותו מידע.

הערות:

1. שימו לב שכלפי חוץ, הרשימה מתנהגת <u>ברשימה מקושרת רגילה של תווים</u>. זה אומר למשל שאורך הרשימה הוא תמיד מספר התווים הכולל ברשימה, ולא מספר ה-nodes שבה (שהוא פרט מימושי בלבד). רשימה שמכילה 5 פעמים W הינה באורך 5 (כיוון שיש בה חמישה תווים) וזאת למרות שמבחינת המימוש, היא מכילה node אחד בלבד (התו W + מספר חזרות 5). החזרת התו באינדקס 3 תחזיר W.
2. הרשימה צריכה תמיד להיות מאוחסנת במספר ה-nodes הקטן ביותר שניתן. למשל, רשימה שמכילה 4 פעמים W צריכה להיות מיוצגת על ידי node אחד בלבד, ולא יכולה להיות מיוצגת על ידי שני nodes שכל אחד מכיל שני W.

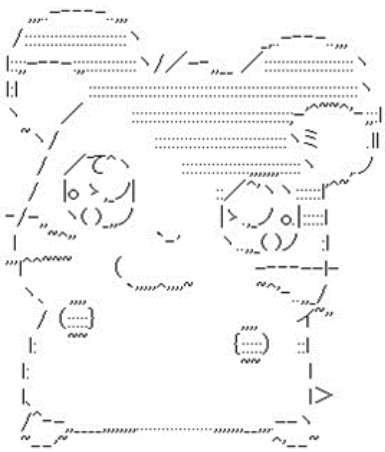
הממשק שתואר בסעיף 3.1 נתון לכם בקובץ הממשק RLEList.h. עליכם לממש את הממשק המתואר בקובץ RLEList.c.

כמו כן, מסופק לכם טסט בסיסי בקובץ RLEList_example_test.c, הנועד ל"בדיקת שפיות" בסיסית ומשמש דוגמה עבור כתיבת הטסטים שלכם.

דגשים נוספים ודרישות מימוש

- **קראו את התיעוד ב-RLEList.h! הוא מגדיר במפורש כל פעולה שעליכם לממש ויעזור לכם במיוחד!**
- קיימות פונקציות שלהן מספר ערכי שגיאה אפשריים. בהערה מעל כל פונקציה בקובץ ist.hRLELI תוכלו למצוא את כל השגיאות שיכולות להתרחש בעת קריאה אליה בקובץ הממשק שמסופק לכם. **במקרה של כמה שגיאות אפשריות החזירו את השגיאה שהוגדרה ראשונה בקובץ.**
  אם מתרחשת שגיאה שאינה ברשימה, יש להחזיר LIST_ERROR_RLE.
- אין הגבלה על מספר האיברים ברשימת RLE.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.
- **בחלק מהפונקציות תידרשו לבצע איחוד ופיצול של חוליות, חשבו מתי כל מקרה נצרך ואיך לממש אותו.**

3.3 *מימוש מערכת לדחיסת AsciiArt*



בחלק זה נממש ממשק לקריאת/כתיבת תמונות מסוג ASCII ART. ASCII ART הוא מונח המשמש לתיאור תמונות המיוצגות על ידי תווי ASCII בלבד, סוג זה של "אומנות" נפוץ כשרוצים להציג תמונות ב-shell. על מנת לחסוך בזיכרון, נשתמש ברשימת ה-RLE שמימשנו בחלק הקודם כדי לשמור את התמונה בצורה דחוסה.
בחלק זה יש לכתוב את קובץ הממשק AsciiArtTool.h עם הפונקציות הבאות, ולממשן בקובץ AsciiArtTool.c:

RLEList asciiArtRead(FILE* in_stream)
הפונקציה קוראת את הקובץ הנתון ודוחסת אותה בשיטת RLE.
פרמטרים:
in_stream – אובייקט מטיפוס *FILE המכיל את התמונה שרוצים לדחוס.
ערך החזרה: רשימה מטיפוס RLEList שמכילה את כל התווים בתמונה.

- **RLEListResult asciiArtPrint(RLEList list, FILE *out_stream)**

הפונקציה כותבת תמונה המיוצגת בעזרת רשימה מטיפוס istRLEL לקובץ.
פרמטרים:
list – רשימה מטיפוס RLEList המכילה כל התווים בתמונה.
out_stream – אובייקט מטיפוס *FILE אליו נכתוב את התמונה.

- **RLEListResult asciiArtPrintEncoded(RLEList list, FILE *out_stream)**

הפונקציה כותבת לקובץ את התמונה בצורה דחוסה (בפורמט שמוחזר מ- **RLEListExportToString**).
פרמטרים:
list – רשימה מטיפוס RLEList שמכילה את התווים בתמונה.
out_stream – אובייקט מטיפוס *FILE אליו נכתוב את התמונה הדחוסה.
ערך החזרה:
RLE_LIST_NULL_ARGUMENT – אם אחד הפרמטרים NULL
RLE_LIST_SUCCESS – במקרה של הצלחה

3.4 תוכנית AsciiArtTool

בחלק זה נממש תוכנית בשם AsciiArtTool שמאפשרת למשתמש לדחוס תמונת ASCII ART. המימוש של פונקציית המain יהיה בקובץ main.c.
התוכנית תקבל דרך ה- command line interface שלושה ארגומנטים (אין ארגומנטים אופציונלים):
flag – מגדיר איזה פעולה לעשות על התמונה. יש 2 flags אפשריים:
e- – לכתוב את התמונה ל-target בצורה מקודדת (encoded)
i- – לכתוב את התמונה בצורה inverted. נסתכל על תמונת ASCII ART כ-inverted אם במקום כל תו @ יש תו רווח, ובמקום כל תו רווח יש תו @ (בכל מופע של @ ורווח בתמונה). התוצאה של פעולה זו אינה תמונה מקודדת.
source – קובץ קלט שמכיל תמונת ASCII ART.
target – קובץ פלט שאליו רוצים לכתוב את התוצאה (הפלט הדחוס או התמונה ה-inverted).
פורמט הפקודה:

- ./AsciiArtTool <flags> <source> <target

לדוגמה:

- ./AsciiArtTool -i dog.txt inverted_dog.txt
- ./AsciiArtTool -e dog.txt encoded_dog.txt

```
 /^ ^\   @@/^@^\
/ 0 0 \  @/@0@0@\
V\ Y /V  @V\@Y@/V
 / - \   @@/@-@\
/     |  @/@@@@|
V__) ||  V__)@||
```

dog.txt inverted_dog.txt

3.5 *דגשים ודרישות מימוש*

- המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Course Material -> Code Conventions.
אי עמידה בכללים אלו תגרור הורדת נקודות.
- על המימוש שלכם להתבצע ללא שגיאות זיכרון (גישות לא חוקיות וכדומה) וללא דליפות זיכרון.

אם בפונקציה מסוימת קיימות מספר אפשרויות לערך שגיאה, אנו נבחר את השגיאה הראשונה על פי סדר השגיאות המופיע תחת הפונקציה הספציפית. השגיאה RLE_LIST_OUT_OF_MEMORY יכולה להתרחש בעת כישלון בהקצאת זיכרון בכל שלב ולא לא משתתפת בעניין הסדר.

- אפשר למצוא ASCII ART נוספים באתר https://www.asciiart.eu/ .
- **המערכת צריכה לעבוד על שרת CSL3.**

3.6 *Makefile*

עליכם לספק Makefile כפי שנלמד בקורס עבור בניית הקוד של תרגיל זה.

- הכלל הראשון ב Makefile יקרא AsciiArtTool ויבנה את התוכנית AsciiArtTool המתוארת למעלה. יש לכתוב את הקובץ כפי שנלמד וללא שכפולי קוד.
- אנו מצפים לראות שלכל מודול קיים כלל אשר בונה עבורו קובץ o., דבר שכפי שלמדתם בקורס - אמור לחסוך הידור של כל התכנית כאשר משנים רק חלק קטן ממנו.
- הוסיפו גם כלל clean.
- תוכלו לבדוק את ה-Makefile שלכם באמצעות הרצת הפקודה make והפעלת קובץ ההרצה שנוצר בסופו.

3.7 *הידור קישור ובדיקה*

התרגיל ייבדק על שרת csl3 ועליו לעבור הידור בעזרת הפקודה הבאה:

```
> gcc -std=c99 -o AsciiArtTool -I/home/mtm/public/2122b/ex1 -Itool -Wall -pedantic-errors -Werror -DNDEBUG *.c tool/*.c
```

פירוט תפקיד כל דגל בפקודת ההידור:

- std=c99 - שימוש בתקן השפה c99.
- o- [program name] – הגדרת שם הקובץ המהודר.
- Wall- – דווח על כל האזהרות.
- pedantic-errors- – דווח על סגנון קוד שאינו עומד בתקן הנבחן כשגיאות.
- Werror- – התייחס לאזהרות כאל שגיאות – משמעות דגל זה שהקוד חייב לעבוד הידור ללא אזהרות ושגיאות.
- DNDEBUG- -מוסיף את השורה #define NDEBUG בתחילת כל יחידת קומפילציה. בפועל מתג זה יגרום לכך שהמאקרו assert לא יופעל בריצת התוכנית.
- *.c קבצי הפתרון שלכם.
- tool/*.c קבצי הפתרון שלכם שנמצאים בתיקיית tool וממשים את המערכת לדחיסת AsciiArt.
- Itool– המטרה של חלק זה היא לכלול את התקייה tool במסלול החיפוש של קבצי ההידור.
- I/home/mtm/public/2122b/ex1 – לכלול את התיקייה במסלול החיפוש של קבצי ההידור. זוהי התיקיית שמכילה את הקבצים המסופקים לכם.

לנוחיותכם, מסופקת לכם תוכנית "בדיקה עצמית" בשם finalCheck, התוכנית בודקת שקובץ ההגשה, קובץ ה-zip, בנוי נכון ומריצה את הטסטים סופקו כפי שירוצו על ידי הבודק האוטומטי. הפעלת התוכנית ע"י הפקודה:

```
~mtm/public/2122b/ex1/finalCheck <submission>.zip
```

הקפידו להריץ את הבדיקה על קובץ ההגשה. אם אתה משנים אותו, הקפידו להריץ את הבדיקה שוב.

3.8 *ולגרינד ודליפות זיכרון*

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. על כן עליכם להשתמש ב-valgrind שמתחקה אחר ריצת התכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך לבדוק האם יש לכם דליפות בתכנית היא באמצעות שתי הפעולות הבאות (שימו לב שחייב להיות main, כי מדובר בהרצה ספציפית):

1. קימפול של השורה לעיל עם הדגל g-.
2. הרצת השורה הבאה:

   valgrind --leak-check=full ./AsciiArtTool

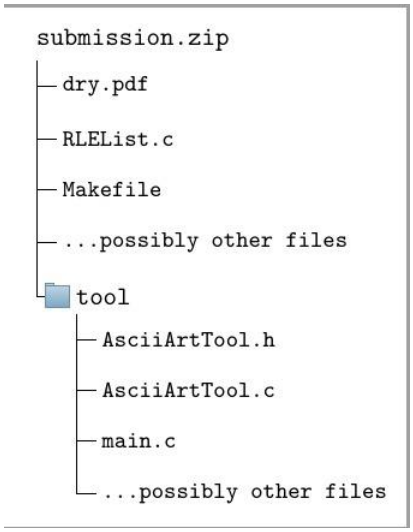
   כאשר AsciiArtTool הוא שם קובץ ההרצה.

הפלט ש-valgrind מפיק אמור לתת לכם, במידה ויש לכם דליפות, את שרשרת הקריאות שהתבצעו וגרמו לדליפה. אתם אמורים באמצעות דיבוג להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התכנית. בנוסף, valgrind מראה דברים נוספים כמו פנייה לא חוקית לזיכרון (segmentation fault) (שלא בוצע בעקבותיה – גם שגיאות אלו עליכם להבין מהיכן הן מגיעות ולתקן.

3.9 *בדיקת התרגיל*

התרגיל ייבדק בדיקה יבשה (מעבר על קונבנציות הקוד והארכיטקטורה) ובדיקה רטובה.
הבדיקה היבשה כוללת מעבר על הקוד ובודקת את איכות הקוד (שכפולי קוד, קוד מבולגן, קוד לא ברור, שימוש בטכניקות תכנות "רעות").
הבדיקה הרטובה כוללת את הידור התכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח בבדיקה שכזו, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות ללא דליפות זיכרון.

4 הגשה



את ההגשה יש לבצע דרך אתר הקורס, תחת
Assignments -> HW1 -> Electronic Submit.

הקפידו על הדברים הבאים:

- יש להגיש את קבצי הקוד וה-makefile מכווצים לקובץ zip (לא פורמט אחר) כאשר כל הקבצים עבור פתרון **המערכת לדחיסת תמונות** מופיעים בתיקיית tools בתוך קובץ ה zip. הקבצים עבור **החלק הראשון** יופיעו בתוך תיקיית השורש.
- יש להגיש קובץ PDF עבור החלק היבש, קראו לקובץ זה בשם dry.pdf ולשים אותו בתיקיית השורש בתוך קובץ ה zip (ליד ה-makefile).
- **אין להגיש אף קובץ מלבד קבצי h וקבצי c אשר כתבתם בעצמכם ואת ה-makefile אשר נדרשתם לעשות ואת ה-PDF של היבש.**
- הקבצים אשר מסופקים לכם יצורפו על ידנו במהלך הבדיקה, וניתן להניח כי הם יימצאו בתיקייה הראשית.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.
- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית שימרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף. כמו כן שימרו עותק של התרגיל על חשבון ה-Google Drive/Github/CSL3 שלכם לפני ההגשה האלקטרונית ואל תשנו אותו לאחריה (שינוי הקובץ יגרור שינוי חתימת העדכון האחרון).
  - כל אמצעי אחר לא יחשב הוכחה לקיום הקוד לפני ההגשה.

בהצלחה !