234124 - מבוא לתכנות מערכות
תרגיל בית 1
סמסטר אביב 2020

תאריך פרסום: 8 באפריל
תאריך הגשה: 12 במאי (עד 23:55)
משקל התרגיל: 12% מהציון הסופי (תקף)
מתרגלים אחראים: אורטל כהן ובר דבש

מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל במודל או בשעות הקבלה. לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.

יש להגיש את תרגיל הבית בזוגות בלבד.

1 הערות כלליות

1. שימו לב: לא תינתנה דחיות במועד הגשת התרגיל פרט למקרים חריגים, ראו נספח 5 בסוף התרגיל. **תכננו את הזמן בהתאם**.

2. שאלות בנוגע לתרגיל ניתן לשאול בפורום הקורס או פרונטלית בסדנאות של המתרגלים **(לא במייל)**. לפני שליחת השאלה - אנא וודאו שהיא לא נענתה כבר ב-F.A.Q או בפורום ושהתשובה אינה ברורה מהדוגמאות והבדיקות שפורסמו עם התרגיל.

3. קראו את התרגיל עד סופו לפני שאתם מתחילים לממש. חובה להתעדכן בעמוד ה- F.A.Q של התרגיל, הכתוב שם מחייב.

4. העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם זאת – מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.

5. מומלץ מאוד לכתוב את הקוד בחלקים קטנים, לקמפל כל חלק בנפרד על השרת, ולבדוק שהוא עובד באמצעות שימוש בטסטים קטנים שתכתבו בעצמכם. לא נדרש מכם בתרגיל להגיש טסטים, אך כידוע, כולנו בני אדם – רצוי לבדוק את התרגיל שלכם היטב כולל מקרי קצה, כי אתם תידרשו לכך.

2 תרגיל יבש (20 נקודות)

2.1 שאלה 1 (10 נקודות) - מיזוג רשימות מקושרות ממוינות

להלן טיפוס Node שמכיל מספר שלם, ערכי שגיאה/הצלחה אפשריים, ושלוש פונקציות:

```
typedef struct node_t {
	int x;
	struct node_t *next;
} *Node;

typedef enum {
	SUCCESS=0,
	MEMORY_ERROR,
	EMPTY_LIST,
	UNSORTED_LIST,
	NULL_ARGUMENT,
} ErrorCode;

int getListLength(Node list);
bool isListSorted(Node list);
ErrorCode mergeSortedLists(Node list1, Node list2, Node *merged_out);
```

ממשו את הפונקציה mergeSortedLists, המקבלת שתי רשימות מקושרות (Linked List) הממוינות בסדר עולה, וממזגת אותן לתוך רשימה מקושרת חדשה הממוינת בסדר עולה. אם כל ההקצאות הצליחו, הפונקציה תחזיר את הרשימה הממוזגת באמצעות הארגומנט merged_out, אשר חייב להיות שונה מ-NULL. בנוסף, הפונקציה תחזיר SUCCESS אם היא סיימה בהצלחה, וערך שגיאה מתאים אם הייתה בעיה בריצת הפונקציה או בקלט שלה. אין לשנות את הרשימות המקוריות.
אם ההקצאה נכשלה, יש להחזיר בארגומנט merged_out את הערך NULL והפונקציה תחזיר MEMORY_ERROR.

לדוגמה, עבור הרשימות (1->4->9) ו-(2->4->8), אם אין שגיאת זיכרון אז הפונקציה תחזיר SUCCESS ותשים ב-merged_out את הרשימה (1->2->4->4->8->9). כדוגמה נוספת, אם הערך של list1 הוא NULL אז הפונקציה תחזיר את הערך EMPTY_LIST.

לנוחותכם, ניתן להשתמש בפונקציות getListLength המחזירה את האורך של רשימה מקושרת, ו-isListSorted המחזירה true אם הרשימה ממוינת או ריקה. אינכם נדרשים לממש את הפונקציות האלה.

כמובן, הימנעו מדליפת זיכרון במימוש של mergeSortedLists.

ניתן להניח כי שתי הרשימות שהפונקציה מקבלת אינן מכילות איבר דמה.

דוגמה לשימוש ב-mergeSortedLists:

```
// left and right are linked lists that were created earlier
Node merged = NULL;
ErrorCode result = mergeSortedLists(left, right, &merged);
```

2.2 שאלה 2 (10 נקודות):

2.2.1 סעיף א (5 נק'):

מצאו 4 שגיאות תכנות ו-4 שגיאות קונבנציה[1] (code conventions) 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.

```
#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;
}
```

2.2.2 סעיף ב (5 נק'):

כתבו גרסה מתוקנת של הפונקציה

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

3 תרגיל רטוב

3.1 חלק א' - מימוש מבנה נתונים (ADT) מילון

מילון (dictionary, לפעמים קרוי גם map) הוא מבנה נתונים שתפקידו למפות *מפתחות לערכים*. בהינתן מפתח כלשהו, ניתן לאחסן במילון ערך עבור מפתח זה, ובשלב מאוחר יותר ניתן לקבל מהמילון את הערך שמופה למפתח זה קודם. פורמלית, מילון מוגדר על ידי הפעולות הבאות:

- put(key, value): ממפה את המפתח key לערך value. במידה ו-key כבר ממופה במילון לערך כלשהו, הערך החדש value מחליף את הערך הקודם.
- get(key): מחזיר את הערך value שממופה ל-key (או NULL אם key לא ממופה לאף ערך)
- contains(key): מחזיר true אם key קיים במילון, כלומר אם ממופה לו ערך כלשהו.

שימו לב שבמילון ניתן למפות לכל מפתח רק ערך אחד. בפרט, אם מפתח כלשהו key כבר קיים במילון, הפעולה put(key,value) מחליפה את הערך הקודם שמופה ל-key בערך החדש. מכאן ברור שלא ייתכן שאותו מפתח מופיע פעמיים במילון. לעומת זאת, ייתכן בהחלט שיהיו כמה מפתחות שונים שממופים לערך זהה.

לדוגמה, מילון עשוי לאחסן מיפוי של מספר קורס (המפתח) לשם הקורס (הערך). כך, בהינתן מספר של קורס נוכל לקבל מהמילון את השם שלו. באופן דומה, מילון יכול למפות מספר קורס (המפתח) למספר הסטודנטים שרשומים בו (הערך). במקרה זה ייתכנו כמה קורסים שממופים לאותו הערך, אם רשומים אליהם אותו מספר סטודנטים.

לשם פשטות, בשאלה זו נתמקד במילונים שבהם גם המפתחות וגם הערכים הינם מחרוזות. להלן דוגמת שימוש במילון כזה לשם מיפוי של תעודת זהות לשם של סטודנט:

```
Map map = mapCreate();
if (map == NULL) {
      return;
}
mapPut(map, "308324772", "John Snow");
mapPut(map, "208364702", "Sansa Stark");
mapPut(map, "308324772", "The Night King");
char* name = mapGet(map, "308324772"); // name = "The Night King";
name = mapGet(map, "208364702"); // name = "Sansa Stark"
bool res = mapContains(map, "108364702") // res = false
res = mapContains(map, "208364702") // res = true
mapDestroy(map);
```

הערה: יש לטפל בערכי החזרה של mapPut ולבדוק האם mapGet החזיר NULL (המצוינים בקובץ map.h שניתן לכם בתרגיל), הטיפול הנדרש בערכים אלו הושמט כדי לשמור על הדוגמה קצרה.

בחלק זה נממש ADT עבור מילון. קובץ הממשק map.h נתון לכם - ונמצא בתיקיית התרגיל. עליכם לכתוב את הקובץ map.c הממשמ את מבנה הנתונים המתואר ב-map.h.

כדי לאפשר למשתמשים במילון (לא לכם!) לעבור על איבריו סדרתית, לכל מילון מוגדר איטרטור (מלשון איטרציה, מעבר על איברים) פנימי ויחיד שבעזרתו יוכל המשתמש לעבור על כל איברי המילון.
כדי לבדוק את התנהגות המילון, מסופק טסט בסיסי בקובץ map_example_test.c.

להלן הגדרת הפעולות על מילון שעליכם לממש:

1. **mapCreate** – יצירת מילון חדש.
2. **mapDestroy** – מחיקת מילון קיים תוך שחרור מסודר של כל הזיכרון שבשימוש.
3. **mapCopy** – העתקת מילון קיים לעותק חדש כולל העתקת האיברים עצמם (מפתחות וערכים).
4. **mapGetSize** – החזרת מספר המפתחות במילון.
5. **mapContains** – יוחזר true אם המפתח הנתון קיים במילון, אחרת יוחזר false.
6. **mapPut** – שינוי ערך של מפתח קיים או הוספת זוג מפתח-ערך חדש למילון.
7. **mapGet** – החזרת הערך הממופה למפתח הנתון.

   הערה: יש להחזיר מצביע למחרוזת השמורה (ולא לעותק).
8. **mapRemove** – מחיקת מפתח מהמילון (יש למחוק גם את הערך השייך למפתח אותו מוחקים).
9. **mapGetFirst** – הזזת האיטרטור לתחילת המילון והחזרת המפתח הראשון.
10. **mapGetNext** – קידום האיטרטור והחזרת המפתח המוצבע על ידו.
11. **mapClear** – ריקון המילון (בשונה מmapDestroy שגם משחרר אותו).

דגשים נוספים ודרישות מימוש:

- **קראו את התיעוד ב-map.h! הוא מגדיר במפורש כל פעולה שעליכם לממש ויעזור לכם במיוחד!**
- קיימות פונקציות להן מספר ערכי שגיאה אפשריים. בהערה מעל כל פונקציה תוכלו למצוא את כל השגיאות שיכולות להתרחש בעת קריאה אליה בקובץ הממשק שסופק לכם (מתחת למילה return בהערה). במקרה של כמה שגיאות אפשריות החזירו את השגיאה שהוגדרה ראשונה בקובץ.
  אם מתרחשת שגיאה שאינה ברשימה, יש להחזיר MAP_ERROR.
- אין הגבלה על מספר האיברים במילון.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.
- במידה ו-mapPut או mapRemove מקבלות NULL כמפתח ו/או כdata, החזירו MAP_NULL_ARGUMENT.
- בתיעוד המופיע ב-map.h עבור חלק מהפונקציות כתוב שהאיטרטור במצב לא מוגדר אחרי הקריאה לפונקציה, המשמעות היא שכאשר איטרטור נמצא במצב זה, אסור למשתמש להניח משהו לגביו, כלומר שאינכם צריכים להבטיח שום דבר בנוגע לערך האיטרטור ואתם יכולים לשנות אותו כרצונכם.

- אם המשתמש קורא ל-mapPut על מפתח שכבר קיים, המידע שקיים אמור להיות מוחלף בערך החדש והפונקציה תחזיר MAP_SUCCESS.
- פונק' mapGet מחזירות את האיבר עצמו (ולא עותק).

3.2 חלק ב' - מימוש מערכת לניהול בחירות

כידוע, בחירות למועצה השבטית בממלכת הדרדסים נערכות אחת ל-4 שנים. נשיא ממלכת הדרדסים, רוצה לייעל את תהליך ההצבעה וספירת הקולות, לכן פנה לחניכי קורס מת"מ בבקשה לעזרה.

תהליך הבחירות בממלכת הדרדסים:

- ממלכת הדרדסים מורכבת מאזורים שונים, כאשר כל דרדס משתייך לאזור מסוים. אין קשר בין שייכות לאזור ובין הצבעה לשבט (כלומר, יכולים להיות שני דרדסים מאותו אזור המצביעים לשבטים שונים ושני דרדסים השייכים לאזורים שונים המצביעים לאותו שבט).
- המועצה השבטית בממלכת הדרדסים מורכבת מנציגי השבטים.

במהלך הבחירות, כל דרדס יכול להצביע לשבט מסוים. בסיום הבחירות, מספר הנציגים מכל שבט נקבע לפי כמות ההצבעות של הדרדסים מהאזורים השונים.

הערות:

מסופק לכם מבנה הנתונים map שכבר מומשו על ידינו. הוסיפו את הקובץ map.h לקבצי ה-h הנדרשים ודאגו שהקובץ libmap.a (שנמצא בתיקיה שסופקה לכם) יימצא בתיקיה הנוכחית. לבסוף קמפלו לפי ההנחיות שבסוף התרגיל – שימו לב לדגלים שנוספו ולהנחיות.

על מנת להוסיף את הקובץ ב-cmake:

1. לפני השורה של add_executable יש להוסיף (שימו לב ל-. בין הסוגריים):

```
link_directories(.)
```

2. לאחר השורה של add_executable יש להוסיף:

```
target_link_libraries(<name_of_executable_file> libmap.a)
```

כאשר name_of_executable_file הוא שם קובץ ההרצה שכתבתם בפקודה add_executable.
אתם רשאים להשתמש בmap שבניתם לצורך מימוש חלק זה, אך זו לא חובה.

3.2.1 טיפוס נתונים ראשי

המערכת הראשית תהיה המבנה **Election**. המבנה מאגד בתוכו את נתוני אזורי ההצבעה, השבטים והצבעות אזרחי הממלכה.

יובהר כי:
- לא יתכן שבמערכת יהיו שני שבטים עם אותו מספר זיהוי (מספר שלם ואי שלילי).
- לא יתכן שבמערכת יהיו שני אזורים עם אותו מספר זיהוי (מספר שלם ואי שלילי).
- יתכן שבט עם אותו מס' זיהוי כמו אזור כלשהו.
- שימו לב שבניגוד למספרי הזיהוי, השמות של השבטים והאזורים אינם ייחודיים, ובפרט יכולים להיות שבטים ו/או אזורים עם אותו שם.

לכל אזור יש מספר פרטים שמייצגים אותו:
- מספר זיהוי של האזור (מספר שלם אי שלילי)
- שם האזור

בנוסף כמובן יש לחשוב על דרך יצירתית (פה התרגיל) לשמור את ההצבעות של אזרחי האזור. יש לתמוך בהוספה של הצבעות (לצרכי מניעת טעויות).
לכל שבט יש לשמור את הנתונים הבאים:
- מספר זיהוי של השבט (מספר שלם אי שלילי).
- שם השבט

3.2.2 פונקציות למימוש:

1. **<u>יצירת מערכת ניהול בחירות</u>**

```
Election electionCreate();
```

הפונקציה תיצור מערכת ניהול בחירות חדשה ללא שבטים או אזורים.

- <u>פרמטרים</u>: אין.
- <u>ערכי שגיאה</u>: NULL במקרה של שגיאה כלשהי, אחרת נחזיר מערכת ניהול בחירות חדשה.

2. **<u>הריסת מערכת ניהול בחירות</u>**

```
void electionDestroy(Election election);
```

הפונקציה תהרוס את מערכת ניהול הבחירות ותשחרר את כל המשאבים שהוקצו לה. במידה והתקבל NULL – אין צורך לבצע דבר.

- <u>פרמטרים</u>: election - המערכת שיש להרוס.
- <u>ערכי שגיאה</u>: הפונקציה לא מחזירה ערך ולכן גם בפרט לא ערך שגיאה כלשהו.

3. <u>הוספת שבט חדש למערכת ניהול הבחירות</u>

```
ElectionResult electionAddTribe(Election election, int tribe_id, const char* tribe_name);
```

הפונקציה תוסיף שבט חדש למערכת ניהול הבחירות.

<u>פרמטרים</u>:
- election - המערכת שאליה נרצה להוסיף שבט חדש.
- tribe_id - מספר זיהוי שבט ייחודי.
- tribe_name - שם השבט. מורכב מאותיות קטנות (lowercase) ורווחים (התו ' ') בלבד (כלומר כל תו אחר הוא תו אסור בשם השבט).

<u>ערכי שגיאה</u>:
- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי השבט הוא מספר שלילי.
- *ELECTION_TRIBE_ALREADY_EXIST* – אם כבר קיים במערכת שבט עם אותו מספר זיהוי.
- *ELECTION_INVALID_NAME* – אם שם השבט מכיל תווים אסורים.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

4. <u>הוספת אזור חדש למערכת ניהול הבחירות</u>

```
ElectionResult electionAddArea(Election election, int area_id, const char* area_name);
```

הפונקציה תוסיף אזור חדש למערכת ניהול הבחירות.

<u>פרמטרים</u>:
- election - המערכת שאליה נרצה להוסיף אזור חדש.
- area_id - מספר זיהוי אזור ייחודי.
- area_name - שם האזור. מורכב מאותיות קטנות (lowercase) ורווחים (התו ' ') בלבד (כלומר כל תו אחר הוא תו אסור בשם האזור).

ערכי שגיאה:

- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי האזור הוא מספר שלילי.
- *ELECTION_AREA_ALREADY_EXIST* – אם כבר קיים במערכת אזור עם אותו מספר זיהוי.
- *ELECTION_INVALID_NAME* – אם שם האזור מכיל תווים אסורים.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

5. קבלת שם שבט מסוים

```
const char* electionGetTribeName (Election election, int tribe_id);
```

הפונקציה תקבל id של שבט ואובייקט מערכת בחירות ותחזיר את שם השבט.

פרמטרים:

- election - המערכת ממנה יש לקבל שם של שבט.
- tribe_id - מספר זיהוי השבט לקבלת השם.

ערכי שגיאה:

- אם התקבל כארגומנט הערך NULL/לא קיים שבט כזה/הקצאת זיכרון כלשהי נכשלה הפונקציה תחזיר NULL.

6. שינוי שם של שבט

```
ElectionResult electionSetTribeName (Election election, int tribe_id,
                                     const char* tribe_name);
```

הפונקציה תשנה את השם של שבט מסוים.

פרמטרים:

- election - המערכת ממנה יש לשנות את השם של השבט.
- tribe_id - מספר זיהוי השבט שיש לשנות את שמו.
- tribe_name - שם השבט. מורכב מאותיות קטנות (lowercase) ורווחים (התו ' ') בלבד.

ערכי שגיאה:

- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי האזור שהתקבל הוא מספר שלילי.
- *ELECTION_TRIBE_NOT_EXIST* - אם לא קיים שבט כזה.
- *ELECTION_INVALID_NAME* – אם שם השבט מכיל תווים אסורים.

- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

7. הסרת שבט ממערכת ניהול הבחירות

```
ElectionResult electionRemoveTribe (Election election, int tribe_id);
```

הסרת שבט ממערכת ניהול הבחירות, למשל עקב פסילה ועדת הבחירות המרכזית. יש למחוק את כל ההצבעות של אזרחי האזורים לשבט זה.

פרמטרים:
- election - המערכת ממנה נרצה להסיר את השבט.
- tribe_id - מספר זיהוי השבט להסרה.

ערכי שגיאה:
- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי השבט שהתקבל הוא מספר שלילי
- *ELECTION_TRIBE_NOT_EXIST* – אם לא קיים שבט כזה.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

8. הסרת אזורים ממערכת ניהול הבחירות

```
ElectionResult electionRemoveAreas(Election election,
                                   AreaConditionFunction should_delete_area);
```

הסרת אזורים ממערכת ניהול הבחירות. יש להסיר את כל האזורים שמספרי הזיהוי מקיימים תנאי מסוים המועבר לפונקציה בתור פרמטר. התנאי מועבר על ידי המצביע לפונקציה should_delete_area המפורט מטה.

הסרת האזורים מתבצעת למשל בעקבות פסילה של קלפיות באזור מחשש לפגיעה בטוהר הבחירות, שיבושים בספירה וכו'. כל ההצבעות של אזרחים מהאזור הנ"ל יימחקו.

פרמטרים:
- election - המערכת ממנה נרצה להסיר אזורים.

- should_delete_area_id - מצביע לפונקציה בוליאנית המקבלת את מספר זיהוי של אזור ומחזירה true אם צריך להסירו ואחרת ומחזירה false.
הגדרת טיפוס המצביע לפונקציה, AreaConditionFunction, מופיע בקובץ election.h.

ערכי שגיאה:
- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

הערה: בקובץ electionTestsExample.c (קובץ שלתוכו תוכלו לכתוב טסטים עבור התרגיל) ישנה דוגמא לשימוש בממשק של הפונקציה הנ"ל כטסט.

9. **הוספת הצבעות של אזרחים מאיזור X לשבט Y במערכת ניהול הבחירות**

```
ElectionResult electionAddVote (Election election, int area_id,
                                int tribe_id, int num_of_votes);
```

הפונקציה תוסיף מספר הצבעות של אזרחים מאזור מסוים לשבט מסוים.

פרמטרים:
- election - המערכת אליה נרצה להוסיף הצבעות.
- area_id - מספר הזיהוי של האזור אליו שייכים האזרחים.
- tribe_id - מספר זיהוי השבט אליה הצביעו האזרחים.
- num_of_votes - מספר ההצבעות שיש להוסיף.

ערכי שגיאה:
- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי של האזור או השבט שלילי.
- *ELECTION_INVALID_VOTES* – אם מספר ההצבעות שיש להוסיף שלילי או אפס.
- *ELECTION_AREA_NOT_EXIST* – מספר הזיהוי של האזור לא קיים במערכת.
- *ELECTION_TRIBE_NOT_EXIST* – מספר הזיהוי של השבט לא קיים במערכת.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

10. הסרת הצבעות של אזרחים מאזור X לשבט Y במערכת ניהול הבחירות

```
ElectionResult electionRemoveVote (Election election, int area_id, int tribe_id,
        int num_of_votes);
```

הפונקציה תסיר מספר הצבעות של אזרחים (אין הכרח שאלו יהיה אזרחים ספציפיים, פשוט מספר ההצבעות ירד) מאזור מסוים לשבט מסוים. (במידה ומספר ההצבעות לאחר ההורדה יהיה קטן מ-0, תוחזר הצלחה אבל המספר יישאר 0)

פרמטרים:

- election - המערכת ממנה נרצה להסיר הצבעות.
- area_id - מספר הזיהוי של האזור אליו שייכים האזרחים.
- tribe_id - מספר זיהוי השבט אליו הצביעו האזרחים.
- num_of_votes - מספר ההצבעות שיש להוריד.

ערכי שגיאה:

- *ELECTION_NULL_ARGUMENT* - אחד הארגומנטים שהתקבל הוא NULL.
- *ELECTION_INVALID_ID* – אם מספר זיהוי של האזור או השבט שלילי.
- *ELECTION_INVALID_VOTES* – אם מספר ההצבעות שיש להוריד שלילי או אפס.
- *ELECTION_AREA_NOT_EXIST* – מספר הזיהוי של האזור לא קיים במערכת.
- *ELECTION_TRIBE_NOT_EXIST* – מספר הזיהוי של השבט לא קיים במערכת.
- *ELECTION_OUT_OF_MEMORY* - מעידה על חוסר בזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.
- *ELECTION_SUCCESS* - במידה והפעולה הצליחה.

11. **מיפוי אזורים לשבטים**

```
Map electionComputeAreasToTribesMapping (Election election);
```

הפונקציה תחזיר מיפוי של אזורים לשבטים, כלומר הפונקציה תחשב עבור כל אזור לאיזה שבט הכי הרבה אזרחים הצביעו.
המap שיוחזר יכיל כמפתחות את מספרי הזיהוי של האזורים (מוצגים כמחרוזת) וכערכים את מספרי הזיהוי של השבטים (מוצגים כמחרוזת).
במידה ויש מספר שבטים אליהם הכי הרבה אזרחים הצביעו, נבחר את השבט עם ה-id הנמוך ביותר.

פרמטרים:

- election – מערכת הבחירות עליה נריץ את החישוב.

ערכי שגיאה:

- במקרה של שגיאה יוחזר NULL, אחרת יוחזר מילון לפי התיאור לעיל.

****שימו לב** שאנחנו מחזירים מילון. במידה ואין אזורים או שבטים במערכת יש להחזיר מילון ריק.

3.2.3 דגשים נוספים ודרישות מימוש

1. המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Code Conventions -> Course Material.
אי עמידה בכללים אלו תגרור הורדת נקודות.
2. על המימוש שלכם לעבור ללא **שגיאות זיכרון** (גישות לא חוקיות וכדומה) וללא **דליפות זיכרון**.
3. במידה ובפונקציה מסוימת קיימות מספר אפשרויות פוטנציאליות לערך שגיאה, אנו נבחר את השגיאה הראשונה על פי סדר השגיאות המופיע תחת הפונקציה הספציפית.
השגיאה ELECTION_OUT_OF_MEMORY
יכולה להתרחש בכישלון בהקצאת זיכרון בכל שלב ולכן לא משתתפת בעניין הסדר.
4. המערכת צריכה לעבוד על שרת csl3.
5. מימוש כל המערכת צריך להיעשות ע"י חלוקה ל-ADT שונים.
נצפה לחלוקה נוחה של המערכת כך שניתן יהיה להכניס שינויים בקלות יחסית ולהשתמש בטיפוסי הנתונים השונים עבור תוכנות דומות.
6. מומלץ לתכנן את ארכיטקטורת המערכת (מבני הנתונים המשתתפים, ה-data types וה-ADT-ים הרלוונטיים) ואת עדכון כל החלקים הרלוונטיים בארכיטקטורה בכל אחת מן הפונקציות. זו הסיבה שהתרגיל הזה נחשב לארוך. הדגש צריך להיות התכנון הנכון.

3.2.4 <u>Makefile</u>

1. עליכם לספק Makefile כמו שנלמד בקורס עבור בניית הקוד של תרגיל זה.
2. הכלל הראשון ב Makefile יקרא election ויבנה קובץ הרצה בשם election – השתמשו בקובץ electionTestsExample.c שסופק לכם, ובמימוש שלכם של הטיפוס Election כפי שתואר מעלה.
3. אנו מצפים לראות שלכל ADT קיים כלל אשר בונה עבורו קובץ .o דבר שכפי שלמדתם בקורס - כדי לחסוך הידור של כל התכנית כאשר משנים רק חלק קטן ממנה.
4. הוסיפו גם כלל clean שמוחק את תוצרי ההידור.
5. על ה-makefile להיות יחיד, ואין להשתמש באלמנטים שלא נלמדו.

הערה: תוכלו לבדוק את ה makefile שלכם באמצעות הרצת הפקודה make והפעלת קובץ ההרצה שנוצר בסופו, כפי שנלמד בתרגול.

3.2.5 הידור ,קישור ובדיקה

התרגיל ייבדק על שרת csl3 ועליו לעבור הידור באופן הבא

עבור ה-map:

```
➢ gcc -std=c99 -o map -Wall -pedantic-errors -Werror -DNDEBUG
  tests/map_example_test.c mtm_map/*.c
```

עבור ה- election:

```
➢ gcc -std=c99 -o election -Wall -pedantic-errors -Werror -DNDEBUG *.c
  tests/electionTests*.c -L. -lmap
```

*.c מציין את כל קבצי c שנמצאים בתיקייה בפרט את הקבצים שמסופקים לכם (אותם אין להגיש).

-lmap מקשר את הקובץ libmap.a שסופק לכם (מניח שהקובץ נמצא בתיקייה הנוכחית).

3.2.6 איתור דליפות זיכרון באמצעות valgrind

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. כדי לוודא זאת, תוכלו להשתמש בכלי בשם valgrind שמתחקה אחר ריצת התכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך להשתמש בכלי על מנת לבדוק האם יש לכם דליפות בתכנית היא באמצעות שתי הפעולות הבאות:

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

```
➢ valgrind --leak-check=full ./election
```

כאשר election זה שם קובץ ההרצה (לא מגישים את קובץ ההרצה לכן תוכלו לתת לו שם כרצונכם).

הפלט ש-valgrind מפיק אמור לתת לכם ,במידה ויש לכם דליפות (והידרתם את התוכנית עם דגל g-), את שרשרת הקריאות שהתבצעו שגרמו לדליפה. אתם אמורים באמצעות ניפוי שגיאות (בדומה לתרגיל 0) להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התכנית. בנוסף, valgrind מראה דברים נוספים כמו קריאה לא חוקית (שלא גררה segmentation fault - גם שגיאות אלו עליכם להבין מהיכן מגיעות ולתקן) בכל מקרה בדיקה שיש בו שגיאת זיכרון כלשהי לא מקבל נקודות.

תוכלו למצוא תיעוד של דגלים נוספים שימושיים של הכלי ע"י man valgrind (או לחפש באינטרנט, לדוג' כאן – תחת Memchek Options).

3.2.7 בדיקת התרגיל

החלק היבש של התרגיל ייבדק ידנית בדומה למבחן. החלק הרטוב של התרגיל ייבדק בדיקה יבשה (מעבר על קונבנציות הקוד והארכיטקטורה) ובדיקה רטובה.

הבדיקה היבשה של המימוש כוללת מעבר על הקוד ובודקת את איכות הקוד (התאמה למסמך הקונוונציות, שכפולי קוד, כתיבת קוד לא נקייה או קוד מבולגן, קוד לא ברור, שימוש בטכניקות תכנות "רעות"). הקוד שאתם מגישים צריך לתאום את הקונוונציות ולהיות מתועד (היכן שצריך - הממשקים

וגם קטעי מימוש שמצריכים זאת), נקי - ללא קטעי קוד לא בשימוש ו/או בהערות, ובנוי לפי כללי התכן הנלמדים.

הבדיקה הרטובה כוללת את הידור התוכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח במקרה בדיקה, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות (זהות ע"י diff) ללא דליפות ושגיאות זיכרון.

4 אופן הגשה

לנוחותכם מסופקת לכם תוכנית "בדיקה עצמית" בשם finalCheck, בתיקיית התרגיל. התוכנית בודקת ש-zip ההגשה בנוי נכון ומריצה את הטסטים שסופקו כפי שיורצו ע"י הבודק האוטומטי. הפעלת התוכנית ע"י:

```
~mtm/public/1920b/ex1/finalCheck <submission>.zip
```

הקפידו להריץ את הבדיקה על קובץ (zip) ההגשה ממש, דהיינו – אם אתם משנים אותו לאחר מכן – הקפידו להריץ את הבדיקה שוב!

4.1 הגשה יבשה

את הפתרון לחלק היבש יש להקליד ולהגיש כקובץ pdf בשם dry.pdf ולשים אותו בתיקייה הראשית של התרגיל (שמגישים בהגשה רטובה). **לא יתקבלו פתרונות בכתב יד ו/או מצולמים** (יקבלו 0 ללא בדיקה). אין להגיש בתא הקורס.

4.2 הגשה רטובה

את ההגשה הרטובה יש לבצע דרך אתר הקורס, תחת

Assignments -> HW1 -> Electronic Submit

פורמט ההגשה כמו באיור משמאל.

```
Submission.zip
├── dry.pdf
├── election.c
├── makefile
├── ... more files
└── mtm_map
    ├── map.c
    └── ... more files
```

- הקובץ **Submission.zip** יכיל את הקבצים הבאים:
  - dry.pdf - פתרון התרגיל היבש.
  - election.c - מימוש החלק הרטוב השני.
  - makefile - קובץ המakefile שהכנתם.
  - קבצים נוספים שמימשתם, אם מימשתם.
  - תיקייה mtm_map שמכילה את:
    - map.c - מימוש החלק הרטוב הראשון.
    - קבצים נוספים שמימשתם, אם מימשתם.

- **שימו לב: השתמשו אך ורק בzip** (פורמט אחר לא יתקבל). אין חשיבות לשם קובץ ה-zip המוגש (Submission.zip מטה).
- הקבצים הבאים מסופקים לכם במהלך הבדיקה **ואין לצרפם להגשה**:
  - election.h, test_utilities.h - תחת התיקייה הראשית.

- map.h - תחת התיקייה mtm_map.
- Libmap.a – תחת התיקייה הראשית. (כדי לקמפל בשרת השתמשו בגירסא של libmap.a שנמצאת בתיקייה: libmap_for_csl3 שבתוך התיקיה הראשית).
- map_example_test.c, electionTestsExample.c - תחת התיקייה tests.

electionTestsExample.c - זהו קובץ שלתוכו תוכלו לכתוב טסטים עבור התרגיל, בדומה לטסטים הכתובים בקובץ map_example_test.c

- הקבצים נמצאים בשרת csl3 תחת התיקייה :

```
~mtm/public/1920b/ex1/
```

- מותר 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-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.