# מבוא לתכנות מערכות
# תרגיל בית 1

## סמסטר אביב 2021

תאריך פרסום: 28.04.2021

תאריך הגשה: 23.05.2021

משקל התרגיל: 12% מהציון הסופי (תקף)

מתרגלים אחראיים: אורטל, מג'ד ומוחמד ג

מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה 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.**

**העדכונים של התרגיל מופיעים ב-highlight צהוב.**

# 1 הערות כלליות

- שימו לב: לא יינתנו דחיות במועד התרגיל פרט למקרים מיוחדים. **תכננו את הזמן בהתאם.**

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

- קראו את התרגיל עד סופו לפני שאתם מתחילים לממש. יתכן שתצטרכו להתאים את המימוש שלכם לחלק עתידי בתרגיל. תכננו את המימוש שלכם לפני שאתם ניגשים לעבוד.

- 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.

- 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 חלק יבש

## 2.1 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,
    UNSORTED_LIST,
    NULL_ARGUMENT,
} ErrorCode;

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

ממשו את הפונקציה mergeSortedLists, המקבלת שתי רשימות מקושרות (Linked List) (שתיהן שונות מ-NULL) וממוינות בסדר עולה, וממזגת אותן לתוך רשימה מקושרת חדשה הממוינת בסדר עולה. הפונקציה תחזיר את הרשימה הממוזגת ואת ערכי השגיאה באמצעות הארגומנט error_code, 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 את הערך NULL והפונקציה תחזיר בארגומנט error_code את הערך MEMORY_ERROR. בנוסף, אם מוחזר קוד שגיאה, השונה מ-SUCCESS, הפונקציה תחזיר את הערך NULL. אין צורך לבדוק אם error_code הינו NULL, במקרה זה לא יוחזר קוד שגיאה.

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

לעזרתכם, ניתן להשתמש בפונקציות getListLength המחזירה את האורך של רשימה מקושרת (אם פונקציה זאת מקבלת NULL היא תחזיר את הערך 0), 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 (איבר שאינו מכיל ערך ונמצא בתחילת רשימה מקושרת. איבר זה מצמצם מקרי קצה).

דוגמה לשימוש ב-mergeSortedLists:

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

# 2.2 מציאת שגיאות

נתונה הפונקציה foo, המקבלת מחרוזת (str) ומצביע ל-int (x). התנהגותה הרצויה של הפונקציה מוגדרת כלהלן:

- הפונקציה מחזירה את כתובת תחילת המחרוזת ההפוכית. הפונקציה לא משנה את המחרוזת המקורית, אלא מקצה מקום חדש, שמה בו את המחרוזת ההופכית, ומחזירה מצביע אליו.
- אם x הוא מצביע תקין, הפונקציה שמה בו את אורך המחרוזת.
- אם מספר התווים במחרוזת אי זוגי, הפונקציה מדפיסה את המחרוזת. אחרת, הפונקציה מדפיסה את המחרוזת ההפוכה. לדוגמא, עבור המחרוזת "hello" הפלט המצופה הוא "hello", ועבור המחרוזת "hi" הפלט המצופה הוא "ih".

בפונקציה זו יש טעויות רבות משני סוגים – שגיאות **נכונות** במימוש הפונקציה הנתונה ו**חריגות מכללי תכנות נכון** שנלמדו בקורס (שאינן משפיעות על התנהגות התוכנית).
עליכם למצוא 8 טעויות בקוד, מהן לפחות 3 מכל סוג.
יש לתקן את הטעויות שציינתם. אתם יכולים להציג תיקון לכל שגיאה, או קוד מתוקן של כל הפונקציה.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* foo(char* str, int* x) {
    char* str2;
    int i;
    x = strlen(str);
    str2 = malloc(*x);
    for (i = 0; i < *x; i++)
        str2[i] = str[*x - i];
    if (*x % 2 == 0) {
        printf("%s", str);
    }
    if (*x % 2 != 0)
    {
        printf("%s", str2);
    }
    return str2;
}
```

# 3 חלק רטוב

## 3.1 מימוש מבנה נתונים גנרי – GDT (Generic Data Type)

בחלק זה נממש ADT גנרי בעבור מילון ממוין. קובץ הממשק map.h נמצא בתיקיית התרגיל. עליכם לכתוב את הקובץ map.c הממשמש את מבנה הנתונים המתואר.

מאחר והמילון הממוין גנרי, יש לאתחל אותו עם מספר מצביעים לפונקציות אשר יגדירו את אופן הטיפול באיברים המאוחסנים בו.

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.

האיטרציה על איברי המילון צריכה להבטיח למשתמש מעבר על האיברים בסדר עולה **מהמפתח הקטן למפתח הגדול** – נדע לעשות זאת באמצעות פונקציית השוואת מפתחות (keyCompare) שמסופקת בעת יצירת המילון. פונקציית compare מחזירה 0 אם שני המפתחות שהיא מקבלת שווים, ערך חיובי אם המפתח הראשון גדול מהשני, וערך שלילי אם המפתח השני גדול מהראשון (בדומה לstrcmp).

כדי לבדוק את התנהגות המילון, מסופק טסט בסיסי בקובץ map_example_test.c.

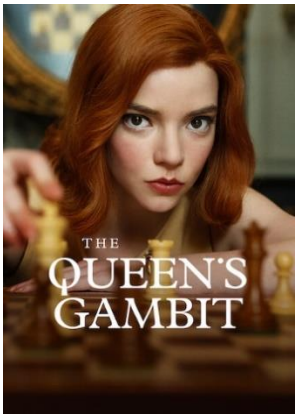
להלן הגדרת הפעולות על מילון שעליכם לממש:

1. **mapCreate** – יצירת מילון ממוין חדש. בפעולה זו מוגדרות לרשימה הפעולות בעזרתן ניתן להעתיק, לשחרר מפתחות וערכים ולהשוות מפתחות.
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 – לא לשכפל אותו (כלומר אין להשתמש בפונקציית העתקת איבר, אלא להחזיר את המצביע לערכו).
8. **mapRemove** – מחיקת איבר מהמילון.
9. **mapGetFirst** – הזזת האיטרטור לתחילת המילון והחזרת עותק של המפתח הראשון.
10. **mapGetNext** – קידום האיטרטור והחזרת עותק של המפתח המוצבע על ידו.
11. **mapClear** – ריקון המילון (בשונה מmapDestroy שגם משחרר את כל הזיכרון שבשימוש המילון).

### 3.1.1 דגשים נוספים ודרישות מימוש

- **קראו את התיעוד ב-map.h! הוא מגדיר במפורש כל פעולה שעליכם לממש ויעזור לכם במיוחד!**
- קיימות פונקציות שלהן מספר ערכי שגיאה אפשריים. בהערה מעל כל פונקציה תוכלו למצוא את כל השגיאות שיכולות להתרחש בעת קריאה אליה בקובץ הממשק שמסופק 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 כ ,dataהחזירו 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 עם מפתח שכבר קיים, המידע שקיים אמור להיות מוחלף בערך החדש והפונקציה תחזיר MAP_SUCCESS.
- הפונקציה mapGet מחזירה את הערך עצמו (ולא עותק).

## 3.2 מימוש מערכת לניהול תחרויות שחמט




לאחר יציאת הסדרה גמביט המלכה, וועד הסטודנטים בטכניון פנו לסטודנטים בקורס מת"ם כדי לעזור בבניית מערכת לניהול תחרויות שחמט בין הסטודנטים בקמפוס.

תחרויות השחמט מתבצעות באמצעות טורנירים בלבד, כלומר לא יכולים להיות משחקי שחמט בין סטודנטים מחוץ לטורניר. כל סטודנט יכול לשחק במספר טורנירים. בנוסף, מספר הטורנירים והמשחקים אינו מוגבל. נתבקשתם לכתוב מערכת היוצרת קבצים המכילים מידע לגבי הטורנירים שנערכו ולגבי השחקנים שהשתתפו במשחקים.

<u>הערות:</u>

מסופק לכם מבנה הנתונים map שכבר מומש 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 השימוש 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 ב-map שבניתם לצורך מימוש חלק זה, אך זה לא חובה.

### 3.2.1 טיפוס נתונים ראשי

המערכת הראשית תהיה המבנה .ChessSystem המבנה מאגד בתוכו את נתוני המשחקים והטורנירים.

יובהר כי:

- לא יתכן שבמערכת יהיו שני טורנירים עם אותו מספר זיהוי (מספר שלם ואי שלילי).
- לא יתכן שבמערכת יהיו שני משחקים עם אותם שחקנים.
- יתכנו שני שחקנים המשחקים במספר טורנירים.

לכל **משחק** יש מספר פרטים שמייצגים אותו:

- מספר תעודות הזהות של השחקנים (אין חשיבות לחלוקת צבעי הכלים לשחקנים).
- תוצאת המשחק.
- משך המשחק בשניות (ניתן להניח כי לא עובר את הגודל המקסימלי של int).

לכל טורניר יש לשמור את הנתונים הבאים:

- מספר זיהוי של הטורניר (מספר שלם אי שלילי).
- מיקום הטורניר.
- משחקים שהתקיימו עבור אותו טורניר.
- זוכה הטורניר.

### 3.2.2 טיפול בשגיאות

במידה ומתרחשת שגיאה על המערכת לפעול כאילו לא בוצעה הפעולה שגרמה לשגיאה, עם סייג של שגיאה אחת, שגיאת זיכרון:

- *CHESS_OUT_OF_MEMORY* - מעידה על כך שנגמר לנו הזיכרון. במקרה כזה יש לשחרר את כל משאבי המערכת שהוקצו עד לנקודה זאת ולסיים את פעולת התכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זיכרון.

במידה ועבור פונקציה מסוימת קיימות מספר אפשרויות פוטנציאליות לערכי שגיאה, נבחר את השגיאה **הראשונה** על פי הסדר של השגיאות המופיע תחת הפונקציה הספציפית. עם זאת קיים סייג יחיד:

- *CHESS_NULL_ARGUMENT* – שגיאה זו היא הראשונה בעדיפות. כלומר, במידה והיא קורה, היא עדיפה על כל שאר השגיאות של הפונקציה. בחלק גדול

מהפונקציות הפרמטרים הנשלחים לפונקציות הם בחלקם מצביעים (מכל סוג). במידה והתקבל מצביע שערכו NULL, השגיאה הזו צריכה להיות מוחזרת.

במידה והפונקציה מחזירה ChessResult והפעולה הצליחה, החזירו *CHESS_SUCCESS*.

### 3.2.3 פונקציות שנדרש לממש

<u>יצירת מערכת ניהול משחקי שחמט</u>

```
ChessSystem chessCreate();
```

הפונקציה תיצור מערכת ניהול משחקי שחמט ללא טורנירים.

- פרמטרים: אין
- ערכי שגיאה: NULL במקרה של שגיאה כלשהי, אחרת נחזיר מערכת ניהול משחקי שחמט.

<u>הריסת מערכת ניהול משחקי שחמט</u>

```
void chessDestroy(ChessSystem chess);
```

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

- פרמטרים: chess – המערכת שיש להרוס.
- ערכי שגיאה: הפונקציה לא מחזירה ערך ולכן גם בפרט לא מוחזר ערך שגיאה כלשהו.

<u>הכנסת טורניר חדש למערכת ניהול משחקי שחמט</u>

```
ChessResult chessAddTournament (ChessSystem chess, int
tournament_id, int max_games_per_player, const char*
tournament_location);
```

הפונקציה תוסיף טורניר חדש למערכת ניהול משחקי שחמט.

- פרמטרים:
  chess – המערכת שאליה נרצה להוסיף טורניר חדש.
  tournament_id - מספר זיהוי טורניר ייחודי, צריך להיות מספר גדול מ-0.
  tournament_location – שם מיקום הטורניר. מורכב מאות גדולה ואחריה מורכב מאותיות קטנות (lowercase) ורווחים (התו ' ') בלבד.
  max_games_per_player – מספר זה מציין את מספר המשחקים המקסימלי בהם שחקן יכול להשתתף בטורניר. מספר זה צריך להיות גדול מ-0.

- ערכי שגיאה:
*CHESS_INVALID_ID* – אם מספר זיהוי הטורניר אינו חוקי.
*CHESS_TOURNAMENT_ALREADY_EXISTS* – אם כבר קיים במערכת טורניר עם אותו מספר זיהוי.
*CHESS_INVALID_LOCATION* – אם שם מיקום הטורניר אינו מקיים את התנאים לעיל.
*CHESS_INVALID_MAX_GAMES* – אם הערך של הפרמטר max_games_per_player אינו תקין.

<u>הכנסת משחק למערכת ניהול משחקי שחמט</u>

ChessResult chessAddGame(ChessSystem chess, int tournament_id, int first_player, int second_player, Winner winner, int play_time);

הפונקציה תוסיף משחק חדש למערכת ניהול משחקי שחמט. ניתן להוסיף משחק בין שני שחקנים ששחקו בעבר בטורניר, אם אחד מן השחקנים נמחק.

- פרמטרים:
chess – המערכת שאליה נרצה להוסיף משחק חדש.
tournament_id – מספר זיהוי של טורניר, צריך להיות מספר גדול מ-0.
first_player – תעודת זהות השחקן הראשון, צריך להיות מספר גדול מ-0.
second_player – תעודת זהות השחקן השני, צריך להיות מספר גדול מ-0.
winner – מכיל ערך של enum המייצג את השחקן שניצח. אם הערך הינו FIRST, אזי השחקן שניצח הינו השחקן הראשון, אם הערך הינו SECOND השחקן המנצח הינו השחקן השני ואם הערך הינו DRAW אזי המשחק הסתיים בתיקו.
play_time – משך המשחק בשניות, זמן זה חייב להיות חיובי, ניתן להניח שערכו קטן מהערך המקסימלי של int.
- ערכי שגיאה:
*CHESS_INVALID_ID* –מספר זיהוי הטורניר או השחקנים אינו חוקי, או ששני השחקנים בעלי אותה ת.ז.
*CHESS_TOURNAMENT_NOT_EXIST* - הטורניר אינו קיים במערכת.
*CHESS_TOURNAMENT_ENDED* – הטורניר הסתיים.
*CHESS_GAME_ALREADY_EXISTS* - אם כבר קיים **בטורניר** משחק עם אותם שני שחקנים.
*CHESS_INVALID_PLAY_TIME* – אם זמן המשחק שלילי.
*CHESS_EXCEEDED_GAMES* – אם השחקן שיחק את מספר המשחקים המקסימלי שניתן לשחק בטורניר.

<u>הסרת טורניר ממערכת ניהול משחקי שחמט</u>

```
ChessResult chessRemoveTournament (ChessSystem chess, int tournament_id);
```

הסרת טורניר ממערכת ניהול משחקי השחמט וכל המשחקים שנערכו בו. יש לעדכן את סטטיסטיקות השחקנים בהתאם.

פרמטרים:
chess – המערכת ממנה נרצה להסיר את הטורניר.
tournament_id – מספר זיהוי טורניר יי חודי להסרה, צריך להיות מספר גדול מ-0.

- ערכי שגיאה:
  *CHESS_INVALID_ID* – אם מספר זיהוי הטורניר שהתקבל אינו מקיים את התנאים לעיל.
  *CHESS_TOURNAMENT_NOT_EXIST* – אם הטורניר אינו קיים.

<u>הסרת שחקן ממערכת ניהול משחקי שחמט</u>

```
ChessResult chessRemovePlayer(ChessSystem chess, int player_id);
```

הפונקציה תסיר את השחקן ממערכת ניהול משחקי שחמט. במשחקים (של כל הטורנירים בהם שיחק ועדיין לא הסתיימו) בהם השתתף השחקן, היריב הינו מנצח אוטומטית לאחר הסרתו. עבור כל משחק שהשחקן השתתף בו, זמן המשחק לא משתנה והמשחק עדיין נחשב כמשחק שהתרחש (ולכן כמות המשחקים באותו טורניר אינו משתנה).

- פרמטרים:
  chess – המערכת ממנה נרצה להסיר את השחקן.
  player_id – מספר זיהוי של שחקן, צריך להיות מספר גדול מ-0.

- ערכי שגיאה:
  *CHESS_INVALID_ID* – אם מספר זיהוי השחקן אינו תקין.
  *CHESS_PLAYER_NOT_EXIST* – אם מספר זיהוי של שחקן אינו קיים במערכת.

<u>סיום טורניר</u>

```
ChessResult chessEndTournament (ChessSystem chess, int tournament_id);
```

הפונקציה תסיים את התחרות ותחשב את תעודת הזהות של מנצח התחרות.
ניקוד תחרות עבור שחקן מחושב באופן הבא, עבור כל משחק בו השחקן שיחק:

- בעת ניצחון במשחק – מתווספות שתי נקודות.
- בעת תיקו – מתווספת נקודה.

- בעת הפסד – לא מתווסף ניקוד כלל.

המנצח בטורניר הינו השחקן בעל הניקוד הגבוה ביותר. אם יש מספר שחקנים בעלי ניקוד גבוה ביותר, יבחר השחקן בעל מספר ההפסדים הנמוך ביותר. אם יש מספר שחקנים כנ"ל עם אותו מספר הפסדים, יבחר השחק בעל מספר הניצחונות הגבוהה ביותר. ולבסוף אם אין אפשרות להחליט על מנצח אחד, יבחר השחקן בעל מספר תעודת הזהות הנמוך ביותר.
לאחר סיום התחרות, לא ניתן להוסיף משחקים עבור אותה תחרות.

- פרמטרים:
  chess – מערכת השחמט עבורה נרצה לסיים טורניר.
  tournament_id – מספר זיהוי טורניר ייחודי לסיום, צריך להיות מספר גדול מ-0.

- ערכי שגיאה:
  *CHESS_INVALID_ID* – אם מספר זיהוי הטורניר אינו תקין.
  *CHESS_TOURNAMENT_NOT_EXIST* – אם מספר זיהוי של טורניר אינו קיים במערכת.
  *CHESS_TOURNAMENT_ENDED*– אם הטורניר כבר הסתיים.

<u>חישוב ממוצע זמני משחק</u>

```
double chessCalculateAveragePlayTime (ChessSystem chess, int
player_id, ChessResult* chess_result);
```

הפונקציה מחזירה את ממוצע זמני משחק (זמני משחקיו בכל הטורנירים) עבור שחקן מסוים.

- פרמטרים:
  chess – המערכת שעבורה נרצה לחשב את ממוצע זמני המשחק.
  player_id – מספר זיהוי של שחקן לחישוב הממוצע, צריך להיות מספר גדול מ-0.
  chess_result – לאחר הרצת הפונקציה, הערך המוצבע יכיל את ערך השגיאה המוחזר.

- ערכי שגיאה:
  *CHESS_INVALID_ID* – אם מספר זיהוי השחקן אינו תקין.
  *CHESS_PLAYER_NOT_EXIST* – אם השחקן אינו קיים במערכת.

שמירת דירוג שחקנים

ChessResult chessSavePlayersLevels (ChessSystem chess, FILE* file);

הפונקציה תחשב את דירוג כל השחקנים במערכת לפי הנוסחה:

$$level = \frac{6 \cdot num\_wins - 10 \cdot num\_losses + 2 \cdot num\_draws}{n}$$

כאשר $num\_wins$ הינו מספר הניצחונות של אותו שחקן, $num\_losses$ הינו מספר הפסדיו, $num\_draws$ הינו מספר המשחקים בהם השתתף ושהסתיימו בתיקו ו-$n$ הינו מספר המשחקים הכולל שהתרחשו במערכת.

השמירה תהיה באמצעות השימוש באובייקט file, כאשר כל שחקן יהיה מיוצג בשורה נפרדת בפורמט:

<id> <level>

רשימה זאת תהיה מסודרת לפי דירוגי השחקנים בסדר יורד. עבור שחקנים באותה דרגה הרשימה תהיה מסודרת לפי תעודות הזהות שלהם בסדר עולה. עבור בדירוג יש להדפיס שתי הספרות העשראניות לאחר הנקודה.

- פרמטרים:
  chess – מערכת השחמט עבורה נרצה להדפיס את דירוג שחקניה.
  במידה ואין שחקנים במערכת יש להשאיר קובץ ריק.
  file – אובייקט מטיפוס *FILE דרכו נשמור את דירוג השחקנים.

- ערכי שגיאה:
  *CHESS_SAVE_FAILURE* – אם התרחשה שגיאה בעת השמירה.

שמירת סטטיסטיקות על טורניר לקובץ

ChessResult chessSaveTournamentStatistics (ChessSystem chess, char* path_file);

הפונקציה תדפיס לקובץ את הסטטיסטיקות עבור כל טורניר שהסתיים. לכל טורניר יודפסו ששת השורות הבאות:

<winner>
<longest game time>
<average game time>
<location>
<number of games>
<number of players>

כאשר:
<winner> - מספר תעודת זהות המנצח במשחק.
<longest game time> - זמן המשחק הארוך ביותר שהתרחש בטורניר.
<average game time> - זמן משחק ממוצע של המשחקים בטורניר, יש להדפיס את שתי הספרות העשרוניות לאחר הנקודה.
<location> - מיקום התרחשות הטורניר.
<number of games> - מספר המשחקים שנערכו באותו טורניר.
<number of players> - מספר השחקנים שהשתתפו בטורניר.

- פרמטרים:
  chess – מערכת השחמט שעבורה נרצה להדפיס את סטטיסטיקות הטורנירים שהסתיימו.
  אם אין טורנירים שהסתיימו במערכת יש להחזיר שגיאה.
  path_file – מסלול הקובץ בתוכו נרצה לשמור את סטטיסטיקות הטורנירים.

- ערכי שגיאה:
  *CHESS_NO_TOURMENTS_ENDED* – אם אין טורנירים שהסתיימו במערכת.
  *CHESS_SAVE_FAILURE* – אם התרחשה שגיאה בעת השמירה.

### 3.2.4 דגשים נוספים ודרישות מימוש

- המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Course -> Code Conventions Material.
  **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 CHESS_SYSTEM_OUT_OF_MEMORY יכולה להתרחש בעת כישלון בהקצאת זיכרון בכל שלב ולא לא משתתפת בעניין הסדר.

- המערכת צריכה לעבוד על שרת CSL3.

- 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.

- מומלץ לתכנן את ארכיטקטורת המערכת (מבני הנתונים המשתתפים, ה-data types וה-ADTים הרלוונטיים) ואת עדכון כל החלקים הרלוונטיים בארכיטקטורה בכל אחת מן הפונקציות. זו הסיבה שהתרגיל נחשב לארוך. הדגש צריך להיות התכנון הנכון.

## Makefile 3.2.5

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

- הכלל הראשון ב Makefile יקרא chess ויבנה את התוכנית chess המתוארת למעלה. יש לכתוב את הקובץ כפי שנלמד וללא שכפולי קוד.
- אנו מצפים לראות שלכל ADT 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.

**הנכם רשאים להשתמש בקובץ exampleMain.c על מנת לספק ל makefile פונקציית main.**

לא חובה להשתמש בקובץ - חשוב ש-make תיצור קבצי o לכל ADT שהשתמשתם בו למערכת.

## 3.2.6 הידור ,קישור ובדיקה

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

```
> gcc -std=c99 -o chessSystem -Wall -pedantic-errors -Werror -DNDEBUG *.c
./tests/chessSystemTestsExample.c  -L. –lmap
```

*.c מציין את כל קבצי c שנמצאים בתיקייה בפרט את הקבצים שמסופקים לכם. **אין להגיש קבצים אלו.**

**מסופקים לכם מימוש של מילון ממוין libmap.a שצריך להעתיק אותו ואת הממשק map.h לתוך תיקיית העבודה. בנוסף מסופקים לכם קבצים לבדיקה ראשונית של החלקים הרטובים בתיקיה tests.**

## 3.2.7 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 ./chessSystem
   כאשר chess זה שם קובץ ההרצה.

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

### 3.2.8 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

## 4.1 הגשה יבשה

את החלק היבש יש להקליד ולהגיש כקובץ pdf בשם dry.pdf ולשים אותו בתיקייה הראשית של התרגיל (שמגישים בהגשה רטובה). ניתן לכתוב בכתב יד ולסרוק, אך רק בתנאי שהכתב והסריקה ברורים. **אין להגיש בתא הקורס**.

## 4.2 הגשה רטובה

את ההגשה הרטובה 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 **החלק השני** מופיעים בתיקיית השורש בתוך קובץ ה zip. הקבצים עבור **החלק הראשון** יופיעו בתוך תיקייה בשם mtm_map, שצריכה להופיע 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-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.