מבוא לתכנות מערכות
תרגיל בית 1

סמסטר חורף 2021-2022

תאריך פרסום: 19/11/2021

תאריך הגשה: 19/12/2021

משקל התרגיל: 12% מהציון הסופי (תקף, מה לעשות)

מתרגלים אחראיים: נלסון גולדנשטיין + רוברט שהלה

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.

עדכונים במסמך התרגיל ביחס לפרסום המקורי מופיעים מודגשים

1 הערות כלליות

- **שימו לב: לא יינתנו דחיות במועד התרגיל פרט למקרים מיוחדים. בנוגע למקרים מיוחדים, ראו נספח 5 בסוף התרגיל. תכננו את הזמן בהתאם.**
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות של אחד מהמתרגלים ונשמח לעזור, או לשאול בפורום של הקורס. לפני שליחת השאלה - אנא וודאו שהיא לא נענתה כבר בפורום ושהתשובה אינה ברורה ממסמך זה, מהדוגמא ומהבדיקות 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- Piazza של התרגיל - הכתוב שם מחייב.
- "המרמה בתוכנה - נענש בחומרה!" (י.ו)

העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם כך – 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.
- הקבצים הנדרשים לתרגיל נמצאים תחת התיקיה "~mtm/public/2122a/ex1/" בשרת csl3.

2 חלק יבש

2.1 זיהוי שגיאות בקוד

2.1.1 סעיף א

מצאו 5 שגיאות תכנות ו-4 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. ניתן להניח שהערך של- times חיובי.

```
#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;
}
```

2.1.2 סעיף ב

כתבו גרסה מתוקנת של הפונקציה.

2.2 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 *mergedOut);
```

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

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 ערכי השגיאה ואת הרשימה הממוזגת באמצעות הארגומנט mergedOut. 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 MEMORY_ERROR והפונקציה תחזיר NULL בארגומנט mergedOut. בנוסף, אם מוחזר קוד שגיאה השונה מ- SUCCESS, הפונקציה תחזיר את הערך NULL בארגומנט.

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

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. <u>אינכם נדרשים לממש את הפונקציות האלה.</u>

כמובן, הימנעו מדליפת זכרון במימוש של mergeSortedLists.

דוגמה לשימוש ב-mergeSortedLists:

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

3 חלק רטוב

לאור כניסתן של ענקיות המסחר לישראל, הוחלט בטכניון ליצור עסק מתחרה בשם מתמיקאה. על הסטודנטים מקורס מת"מ הוטל לבנות את המערכת הממוחשבת ששולטת במלאי המוצרים הקיימים, כולל היכולת להוסיף מוצרים חדשים, להסיר מוצרים, ליצור ולשלח הזמנות מחוץ למחסן וכו'.

אנו נבנה את המערכת בשני שלבים. בשלב הראשון נבנה מבנה נתונים Amount Set, בדומה למבני הנתונים שלמדנו בכיתה, עבור מחרוזות. בשלב השני נשתמש במבנה הנתונים בגרסתו הגנרית שמסופק לכם כדי לבנות את מערכת מתמיקאה.

1.1 מימוש מבנה נתונים – Amount Set

בחלק זה נממש ADT עבור "סט כמותי" – סט של איברים יחודיים, אשר בנוסף מחזיק לכל איבר את הכמות שלו בסט. הממשק של הסט נמצא בקובץ amount_set_str.h. עליכם לכתוב את הקובץ amount_set_str.c, אשר מממש את מבנה הנתונים המתואר עבור מחרוזות.

כדי לאפשר למשתמשים בסט (לא לכם!) לעבור על איבריו סדרתית, לכל סט מוגדר איטרטור (מלשון איטרציה, מעבר על איברים) פנימי ויחיד שבעזרתו יוכל המשתמש לעבור על כל איברי הסט. האיטרציה על איברי הסט צריכה להבטיח למשתמש מעבר על האיברים בסדר עולה מאיבר הקטן לאיבר הגדול – נדע לעשות זאת לפי הסדר הלקסיקוגרפי באמצעות פונקציית השוואת האיברים (strcmp).

1.1.1 פונקציות שנדרש לממש

להלן תיאור קצר של הפונקציות אותן אתם נדרשים לממש. פירוט נוסף, כולל משמעות הפרמטרים וערכי החזרה האפשריים של כל פונקציה, נמצא בקובץ amount_set_str.h. על המימוש שתספקו לענות על דרישות כל התיעוד (המובא כאן ובקובץ ה-h) ויש להניח שכל דרישות התיעוד ייבדקו.

1.1.1.1 יצירת סט חדש

```
AmountSet asCreate();
```

יוצר סט חדש ריק.

- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.2 הריסת סט קיים

```
AmountSet asDestroy(AmountSet set);
```

הורס סט ואת כל האיברים שהוא מכיל.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.3 העתקת סט

```
AmountSet asCopy(AmountSet set);
```

יוצר סט חדש, ומעתיק לתוכו את כל האיברים מ-set ואת הכמות של כל איבר.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.4 *החזרת מספר האיברים בסט*

```
int asGetSize(AmountSet set);
```

כל איבר נספר פעם אחת, ללא תלות בכמות שלו בסט.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.5 *בדיקה האם איבר נמצא בסט*

```
bool asContains(AmountSet set, const char* element);
```

מחזיר true אם element נמצא ב-set, ו-false אחרת.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.6 *החזרת הכמות של איבר בסט*

```
AmountSetResult asGetAmount(AmountSet set, const char* element, double
*outAmount);
```

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.7 *רישום איבר חדש לסט*

```
AmountSetResult asRegister(AmountSet set, const char* element);
```

מכניס איבר חדש לסט, עם הכמות 0. כלומר, לאחר פעולה זו, האיבר החדש "נמצא בסט עם כמות אפס ".

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.8 *שינוי הכמות של איבר קיים בסט*

```
AmountSetResult asChangeAmount (AmountSet set, const char* element, double
amount);
```

משנה את הכמות של איבר בסט. אם amount חיובי אז מגדיל, אם שלילי אז מקטין, ואם amount הוא אפס אז אין שינוי. לא ניתן לבצע פעולה זו עבור איבר שטרם הוכנס לסט או שכבר הוצא מהסט.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

הבהרות:

- אפילו אם amount הוא אפס, עדיין צריך להחזיר AS_ITEM_DOES_NOT_EXIST אם element לא קיים ב-set ו-AS_SUCCESS אם element כן קיים ב-set.
- אסור שכמות של איבר בסט תהיה שלילית. אם amount הוא שלילי וביצוע הפעולה יגרום לכמות של האיבר בסט לרדת מתחת לאפס, אז הפעולה צריכה להיכשל ולהחזיר AS_INSUFFICIENT_AMOUNT.

1.1.1.9 *מחיקת איבר מהסט*

```
AmountSetResult asDelete(AmountSet set, const char* element);
```

מוציא איבר קיים מהסט ומוחק את האיבר. כלומר, לאחר פעולה זו, האיבר כבר לא נמצא בסט, אפילו לא "עם כמות אפס".

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.10 ריקון הסט

```
AmountSetResult asClear(AmountSet set);
```

מוחק את כל האיברים מהסט. כלומר, לאחר פעולה זו, הסט נמצא במצב זהה למצבו מיד אחרי היצירה הראשונית שלו.

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.11 הזזת האיטרטור לתחילת הסט והחזרת האיבר הראשון

```
char* asGetFirst(AmountSet set);
```

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.1.12 קידום האיטרטור והחזרת האיבר המוצבע על ידו

```
char* asGetNext(AmountSet set);
```

- פרמטרים: ראו קובץ amount_set_str.h
- ערכי חזרה: ראו קובץ amount_set_str.h

1.1.2 הכנת טסטים (10 נקודות בונוס)

- לכל פונקציה מהפונקציות הנ"ל, אתם נדרשים לספק טסטים מתאימים שבודקים את נכונותה ומקרי קצה. הציון יינתן לפי כיסוי מקרי הקצה ע"י הטסטים וגם כיסוי הפונקציונלית של ה- Amount Set. אתם נדרשים לספק את הטסטים בקובץ amount_set_str_main.c שיכיל את כל הטסטים שלכם, יחד עם ההרצה שלהם.
- אתם נדרשים לספק את הטסטים בקובץ amount_set_str_tests.c שיכיל את מימוש הטסטים, וקובץ amount_set_str_tests.h שמכיל את ההכרזות של פונקציות הטסטים.
- עליכם לכלול פונקציית main שמריצה את הטסטים שלכם בקובץ amount_set_str_main.c.
- אתם יכולים לראות כדוגמה את הטסטים המסופקים לכם למערכת matamikya.

1.1.3 דרישות והערות נוספות

- אתם נדרשים לממש את Amount Set עם רשימה מקושרת (שאתם ממשים בעצמכם).
- עבור מימוש ה-Amount Set, מותר להשתמש רק בספריה הסטנדרטית של C. בפרט, אין להשתמש ב-set ו-list שמומשו ע"י סגל הקורס.
- לכל פונקציה המחזירה AmountSetResult מפורטים ערכי השגיאה האפשריים לה. במקרה שפונקציה מקבלת קלט שמתאימים לו יותר מערך שגיאה אחד, יש להחזיר את ערך השגיאה הראשון כפי שמופיע בתיעוד הפונקציה בקובץ ה-h.
- עליכם להניח כי לא יתכנו מקרי שגיאה פרט לאלו המצוינים בתיעוד של כל פונקציה. בניגוד לאמור לעיל עבור שגיאה המעידה על בעיית זיכרון יש להחזיר מכל פונקציה בה היא מתרחשת את הערך AS_OUT_OF_MEMORY אם ערך החזרה שלה הוא מטיפוס AmountSetResult, או NULL אם ערך החזרה שלה הוא מצביע.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.
- במקרה של שגיאה, מבנה הנתונים ביציאה מהפונקציה צריך להיות זהה למצבו בכניסה לפונקציה.
- אין הגבלה על מספר האיברים בסט.
- בהתאם לתיעוד בקובץ ה-h, ישנן פונקציות שאחרי הקריאה להן מצב האיטרטור לא מוגדר, או שלא מצוין מצבו. אם לא מצוין מצב האיטרטור, המשמעות היא שמצב האיטרטור אחרי הקריאה לפונקציה הוא לא מוגדר. כאשר איטרטור נמצא במצב זה, זה אומר שאסור למשתמש להניח משהו עליו, כלומר

שאינכם צריכים להבטיח שום דבר בנוגע לערך האיטרטור ואתם יכולים לשנות אותו כרצונכם. זה בא בשביל להקל עליכם. שימו לב, האיטרטור נמצא במצב מוגדר רק בסוף פונקציות עבורן מצוין בתיעוד במפורש שזה המצב.

- שימו לב להבדל בין הכנסה של איבר חדש לסט לבין הוספה של איבר קיים לסט – הכנסה של איבר חדש גורמת לסט "להכיר" את האיבר הזה, אבל בהוספת איבר קיים ניתן אך ורק לשנות את הכמות של איבר שהסט כבר מכיר".
- באופן דומה, שימו לב להבדל בין מחיקה של איבר לבין שינוי הכמות של איבר – מחיקה גורמת לסט "לשכוח" שהאיבר היה קיים אי פעם, אבל הוספת כמות שלילית לא יכולה לגרום לסט "לשכוח" את האיבר, אפילו אם הכמות שלו יורדת לאפס.
- אתם יכולים להוסיף עוד קבצים במימוש שלכם, כל הקבצים שאתם מוסיפים יהיו בעלי שמות המתחילים ב-"amount_set_str".

1.2 מימוש מערכת לניהול מחסן מוצרים – מתמיקאה

1.2.1 טיפוס נתונים ראשי

המערכת מרוכזת תחת טיפוס נתונים בשם Matamikya, להלן המחסן שלנו. המחסן נדרש להיות בעל מספר יכולות אשר יפורטו כעת:

ניהול מלאי המחסן: למחסן זה ניתן להוסיף מוצרים שונים. המוצרים מזוהים על ידי מספר פרמטרים:

- שם מוצר מטיפוס מחרוזת
- מספר מזהה מוצר ייחודי לו id. (אי שלילי).
- מידע מותאם אישית מטיפוס MtmProductData אשר מוצהר בmatamikya.h ומחזיק מידע רלוונטי נוסף על כל מוצר. לדוגמא המחיר ליחידת מידה עבור ירקות הוא שקל לקילוגרם ואילו עבור מכשירי חשמל המחיר הוא ליחידה שלמה אחת.
- יחידת המידה של המוצר. כל מוצר הוא בעל יחידות שלמות, חצי-שלמות או רציפות. לדוגמה, טלוויזיה ניתן למכור רק ביחידות שלמות, אבטיח ניתן למכור בשלמים או בחצאים, מלפפונים ניתן למכור לפי משקל כלשהו (יחידה רציפה). יחידות המידה המותרות הן היחידות המופיעות ב-MatamikyaAmountType. יחידת המידה של מוצר קובעת מהן הכמויות המותרות של המוצר במחסן ובהזמנה (ראו פירוט בקובץ ה-h).
- באופן כללי טיפוס זה הוא למימושכם ויכול להחזיק כל מידע שתמצאו רלוונטי להחזיק לצורכי התרגיל.

הבהרה: יצוין פה כי לא תיתכן כפילות של מוצרים במחסן. כלומר במידה ואנו מוכרים תפוחים במחסן ייתכן כי מלאי התפוחים ירד ויעלה לאורך התוכנית, אך לא ייתכן מצב בו יש יותר ממוצר אחד המזוהה על ידי id המתאים לתפוחים.

ניהול הזמנות:

חלק נוסף במחסן שלנו הוא היכולת לקבל הזמנות מלקוחות. ההזמנות עשויות להכיל מספר מוצרים מכמויות שונות. כל הזמנה מזוהה על ידי מספר מזהה ייחודי לה מטיפוס unsigned int אשר יבדיל אותה משאר ההזמנות הקיימות במערכת. ביצוע הזמנה מורכב מ-3 שלבים אשר לא מתחייב שיתרחשו בצמוד:

1. יצירת הזמנה חדשה.
2. הוספת מוצרים להזמנה לפי בחירת הלקוח.
3. שילוח ההזמנה.

הבהרות בנוגע לניהול ההזמנות:

- יצירת הזמנה אינה אומרת שילוח הזמנה. כפי שאתם מוסיפים מוצרים ל"סל" בעת הזמנות באינטרנט הסל שומר על מצבו עד אשר תחליטו לבצע check out. **יתכן מצב בו קיימים בהזמנה מוצרים שאזלו מן המלאי או שנוסיף להזמנה מוצר מכמות שגדולה מהכמות הקיימת במחסן. עם זאת לא ייתכן מצב שנוסיף להזמנה מוצר שהמחסן "לא מכיר" כלומר אינו קיים בו כלל.** לפירוט המלא ראו את התיעוד בקובץ matamikya.h
- לא ייתכן מצב בו לשתי הזמנות אותו מספר מזהה.
- **ייתכנו מספר הזמנות הקיימות בו זמנית במערכת**. חשבו איזה מבנה נתונים מתאים על מנת לקיים את הדרישה הזו.

ניהול הכנסות:

לאחר ששולחה הזמנה (ראו פירוט על פונקצית mtmShipOrder בהמשך) המחסן בעצם "עשה רווח", להלן **"הכנסה"**. אם לדוגמא טלוויזיה עולה 500, ולקוח יצר הזמנה עם 2 טלוויזיות אז לאחר שילוח ההזמנה (mtmShipOrder) המחסן עשה **"הכנסה"** של 1000 עבור המוצר הזה . יש לעקוב אחר ההכנסות של המוצרים השונים לאורך התכנית. כלומר מהזמנה להזמנה ההכנסות ממוצר מסוים יכולות לגדול.

הבהרות בנוגע לניהול ההכנסות:

- שימו לב שלא יתכן מצב בו ההכנסות קטנות, אלא רק גדלות.
- ייתכן מצב בו היו הכנסות ממוצר מסוים ולאחר מכן המוצר הוסר לחלוטין מהמחסן. במצב שכזה אין צורך לשמור את התיעוד שהיה על ההכנסות ממוצר זה.

הבהרה: בחלק 3.2 ניתן להשתמש בlist ו- set שסופקו לכם על ידי סגל הקורס בקובץ libmtm.a ואנו ממליצים לכם גם לעשות שימוש בamount_set הגנרי המסופק לכם בקובץ libas.a.

1.2.2 פונקציות ומבני נתונים שנדרש לממש

להלן תיאור קצר של הפונקציות אותן אתם נדרשים לממש. פירוט נוסף, כולל משמעות הפרמטרים וערכי החזרה האפשריים של כל פונקציה, נמצא בקובץ matamikya.h. על המימוש שתספקו לענות על דרישות כל התיעוד (המובא כאן ובקובץ ה-h) ויש להניח שכל דרישות התיעוד ייבדקו.

1.2.2.1 מבני נתונים וטיפוסים נוספים

Matamikya – מצביע לטיפוס מבנה הנתונים כפי שמוסבר בחלק הקודם.

MtmProductData – מצביע למידע "מותאם" למוצר.

(MtmProductData)(*MtmCopyData) MtmProductData – מצביע לטיפוס של פונקציה המיועדת להעתקת המידע המותאם של כל מוצר.

(MtmProductData)(*MtmFreeData) void – מצביע לטיפוס של פונקציה המיועדת לשחרור המידע המותאם של כל מוצר.

(MtmProductData, const double amount)(*MtmGetProductPrice) double – מצביע לטיפוס של פונקציה המיועדת בהינתן כמות רצויה amount ומידע על מוצר MtmProductData להחזיר את המחיר של המוצר ביחס לכמות הרצויה. ניתן להניח ש-amount הוא בעל ערך תקין למוצר.

הערה מנחה לגבי הטיפוסים והפונקציות עד כה: בניגוד לדוגמאות שראיתם בתרגול ובהרצאה, שם הממשק הגנרי היה חלק מהכלי אותו מימשנו בלבד (לדוגמא מחסנית גנרית בתרגול), עבור תרגיל זה ישנה רמה נוספת של גנריות. לדוגמא, במקום שלמחסן שלנו תהיה פונקציה יחידה אשר מחזירה לנו את המחיר עבור כל מוצר במחסן. על המשתמש לספק את הפונקציה הזו לכל מוצר ומוצר כי למוצרים שונים ייתכן חישוב שונה של המחיר (לדוגמא בהינתן מבצעים, הנחות וכו').

הפונקציות הבאות אינן דורשות כל גנריות ועליכם לכתוב אותה בצורה שהכי נוחה לכם.

1.2.2.2 יצירת מחסן

```
Matamikya matamikyaCreate();
```

פונקציה היוצרת את המחסן שלנו ומחזירה את טיפוס הנתונים המדובר matamikya.

1.2.2.3 הריסת מחסן

```
void matamikyaDestroy(Matamikya matamikya);
```

פונקציה שבהינתן מחסן מטיפוס Matamikya משחררת את כל הזיכרון שהוקצה עד כה לטובת בניית מחסן זה. יש לוודא שאין זליגות זיכרון כמוסבר בהמשך.

1.2.2.4 רישום מוצר חדש למחסן

```
MatamikyaResult mtmNewProduct(Matamikya matamikya, const unsigned int id,
const char *name, const double amount, const MatamikyaAmountType
amountType, const MtmProductData customData, MtmCopyData CopyData,
MtmFreeData FreeData,  MtmGetProductPrice ProdPrice);
```

פונקציה שבהינתן כל המידע הדרוש על מוצר חדש יוצרת את המוצר ומכניסה אותו למחסן. אם אחד הפרמטרים לא תקין, יש להחזיר ערך שגיאה מתאים.

אם אחד מהארגומנטים שהוא מצביע הוא NULL יש להחזיר MATAMIKYA_NULL_ARGUMENT.

1.2.2.5 שינוי המלאי של מוצר במחסן

```
MatamikyaResult mtmChangeProductAmount(Matamikya matamikya, const unsigned int id, const double amount);
```

פונקציה אשר תפקידה בהינתן מחסן, מזהה מוצר **הקיים במחסן** וכמות, להוסיף/להחסיר את הכמות שהועברה כארגומנט למלאי הקיים במחסן. אם הכמות שהועברה (amount) היא מספר חיובי יש להוסיף את הכמות הזו למלאי, אם הכמות שלילית יש להחסיר את הכמות הזו ממלאי המחסן, ואם הכמות שהועברה היא 0 אין לעשות דבר וזה תקין.

הבהרה: אם אחד הפרמטרים לא תקין, יש להחזיר ערך שגיאה (לא MATAMIKYA_SUCCESS) אפילו אם amount הוא אפס.

1.2.2.6 מחיקת מוצר מהמחסן

```
MatamikyaResult mtmClearProduct(Matamikya matamikya, const unsigned int id);
```

פונקציה אשר תפקידה, בהינתן מחסן ומזהה מוצר, להסיר את המוצר בעל המזהה שהועבר כארגומנט מן המחסן לחלוטין. המוצר יימחק והזיכרון שתפס במחסן ישוחרר. בנוסף, יש למחוק את המוצר מכל הזמנה קיימת שבה הוא מופיע.

1.2.2.7 יצירת הזמנה חדשה

```
unsigned int mtmCreateNewOrder(Matamikya matamikya);
```

פונקציה אשר תפקידה הוא ליצור הזמנה חדשה **ריקה**. הפונקציה תחזיר מספר מטיפוס unsigned int אשר יהווה את המספר המזהה הייחודי עבור הזמנה זו ואשר יבדיל אותה מכל שאר ההזמנות אשר קיימות במערכת. ניתן להניח שכמות ההזמנות שיבוצעו במהלך ריצת התוכנית קטן מהערך המקסימלי הניתן לייצוג ע"י טיפוס unsigned int. **יובהר פה שוב כי ייתכן שבמחסן יהיו מספר הזמנות במקביל. לדוגמא, לאחר 2 קריאות רצופות לפונקציה זו ייוצרו 2 הזמנות ריקות בעלי מספר מזהה שונה**

1.2.2.8 שינוי כמות מוצר בהזמנה

```
MatamikyaResult mtmChangeProductAmountInOrder(Matamikya, const unsigned int orderId, const unsigned int productId, cosnt double amount);
```

פונקציה אשר בהינתן מחסן, מספר הזמנה קיימת, מספר מוצר וכמות רצויה מן המוצר, תוסיף/תחסיר את הכמות שהועברה **אם המוצר כבר קיים בהזמנה**. כמקודם אם הכמות גדולה מאפס היא תתווסף למוצר בהזמנה ואם היא קטנה מאפס היא תוחסר מן המוצר שבהזמנה (עבור העברה של כמות 0 כמות המוצר בהזמנה נשארת ללא שינוי). שימו לב כי ייתכן מצב בו נוסיף להזמנה מוצר שאזל מן המלאי, אך לא מוצר שכבר נמחק או לא היה קיים מעולם במחסן.

הבהרה: אם אחד הפרמטרים לא תקין, יש להחזיר ערך שגיאה (לא MATAMIKYA_SUCCESS) אפילו אם amount הוא אפס.

הערה חשובה:

<u>יש לשים לב כי יצירת הזמנה של מוצר Y עם כמות X אין משמעותה שיש באותו הרגע להפחית את הכמות X מן המלאי הקיים במחסן של המוצר Y.</u>

1.2.2.9 שילוח הזמנה

```
MatamikyaResult mtmShipOrder(Matamikya matamikya, const unsigned int orderId);
```

פונקציה אשר בהינתן מחסן ומספר הזמנה תוציא הזמנה אל הפועל. כלומר במידה וכל המלאי הנדרש להזמנה המדוברת קיים במחסן, הכמויות הדרושות יוחסרו ממלאי המחסן וההזמנה תמחק ממאגר ההזמנות הקיימות במערכת.

1.2.2.10 ביטול הזמנה

```
MatamikyaResult mtmCancelOrder(Matamikya matamikya, const unsigned int
orderId);
```

בהינתן מחסן ומספר הזמנה, תבוטל ההזמנה המדוברת, ותוסר ממאגר ההזמנות הקיימות במערכת.

1.2.2.11 הדפסת מלאי המוצרים במחסן

```
MatamikyaResult mtmPrintInventory(Matamikya matamikya, FILE *output);
```

בהינתן מחסן וקובץ פלט יודפס כל מלאי המחסן אל הקובץ לפי הפורמט הבא. תחילה יש להדפיס את הכותרת "Inventory Status:" ולאחר מכן יש להדפיס את פרטי המוצרים הקיימים במחסן ממוינים בסדר עולה לפי מספר המזהה – id – בשורות נפרדות. אין להוסיף רווחים מיותרים בסוף שורה.

Inventory Status:
name: <name>, id: <id>, amount: <existing amount>, price: <price per unit>

השורה הראשונה היא הכותרת. השורה השנייה היא פורמט ההדפסה לכל מוצר. כלומר לכל מוצר הקיים במחסן יש להדפיס שורה המתאימה לו. **שימו לב עבור ההדפסה של כל שורת מוצר עליכם לעשות שימוש בפונקציה printProductFormat שסופקה לכם.** id הוא מזהה המוצר הייחודי לו, name הוא שם המוצר, amount הוא כמות המוצר הקיימת במלאי המחסן ו-price הוא המחיר של המוצר ליחידת מדידה. לדוגמא מחיר טלוויזיה הוא עבור יחידה אחת ומחיר תפוחים הוא עבור קילוגרם.

דגש: עבור מחסן קיים שאינו מכיל מוצרים יש להדפיס את הכותרת ובזאת לסיים את ההדפסה כי אין מוצרים להדפיס.

דוגמה:

Inventory Status:
name: Apple, id: 0, amount: 205.4, price: 10.9
name: TV, id: 15, amount: 51, price: 1299

1.2.2.12 הדפסת הזמנה

```
MatamikyaResult mtmPrintOrder(Matamikya matamikya, const unsigned int
orderId, FILE *output);
```

בהינתן מחסן, מספר הזמנה וקובץ פלט תודפס ההזמנה אל קובץ הפלט לפי הפורמט הבא.

בהדפסת הזמנה יש תחילה להדפיס כותרת המזהה את ההזמנה המודפסת לפי ה-id המתאים לה. לאחר מכן בשורות נפרדות יש להדפיס את תוכן ההזמנה, כלומר המוצרים המרכיבים אותה ממוינים בסדר עולה לפי מספר המזהה – id. לבסוף יש להדפיס "שורת סיכום".

Order <order id> Details:
name: <name>, id: <id>, amount: <amount in order>, price: <price for amount>

Total Price: <total price for the entire order>

פורמט השורה למוצר זהה לקודם פרט לכך שכעת amount מפרט את כמות המוצר בהזמנה זו. price כעת מתאר את המחיר למוצר עבור הכמות שנמצאת בהזמנה. בסוף הדפסת המוצרים יש להדפיס את "שורת הסיכום". שורה זו מורכבת מ-2 שורות נפרדות: בראשונה יש להדפיס 10 תווי '-' . בשנייה יש להדפיס לפי הפורמט הנראה למעלה את סך העלות הכוללת עבור הזמנה זו.

דגש: במידה וההזמנה קיימת אך אין בה מוצרים יש להדפיס את הכותרת ואת "שורת הסיכום באופן מתאים, כלומר עלות הזמנה שכזו המכילה 0 מוצרים היא 0.**שימו לב עבור ההדפסה של הכותרת , שורות המוצר ו"שורת הסיכום עליכם לעשות שימוש בפונקציות printOrderHeading, printProductFormat, printOrderSummary בהתאמה שסופקו לכם.**

דוגמה להדפסת הזמנה:

Order 3 Details:
name: Apple, id: 0, amount: 1.4, price: 15.26
name: TV, id: 15, amount: 2, price: 2598

Total Price: 2613.26

1.2.2.13 *הדפסת המוצר המכניס ביותר*

```
MatamikyaResult mtmPrintBestSelling(Matamikya matamikya, FILE *output);
```

פונקציה אשר בהינתן מחסן וקובץ פלט מדפיסה לקובץ הפלט את המוצר ה"מכניס" ביותר. הכוונה במכניס היא סה"כ ההכנסות שנצברו משילוח של המוצר בהזמנות מתחילת התוכנית. ההדפסה תעשה לפי הפורמט הבא. כותרת מתאימה ולאחר מכן הדפסת המוצר.

Best Selling Product:
name: <name>, id: <id>, total income: <total income from that product>

בפורמט השורה יודפס השם וה-id של המוצר, הפרמטר האחרון שיודפס הוא סך ההכנסה ממוצר זה מאז שהמוצר נוסף למחסן. **שימו לב עבור הדפסה זו עליכם לעשות שימוש בפונקציה mtmPrintIncomeLine שסופקה לכם**

דגש: במידה ושני מוצרים הם בעלי אותה הכנסה ה-id של המוצרים ישמש כשובר שיוון. יודפס המוצר עם ה-id הקטן יותר.

בנוסף עבור מחסן שטרם עשה מכירות כלשהן אין הכנסה לאף מוצר, לכן במצב זה הפונקציה תדפיס:

Best Selling Product:
none

1.2.3 דרישות והערות נוספות

- הפונקציות ומבני הנתונים אותם אתם נדרשים לממש נמצאים ומפורטים לרמת קלט פלט בקובץ matamikya.h שסופק לכם.
  מסופקים לכם גם מבני הנתונים set ו-list שכבר מומשו על ידי סגל הקורס, וקבצי ה-o שלהם מוכלים בקובץ libmtm.a. בשביל להשתמש בהם הוסיפו #include לקובץ matamikya.c. יש לדאוג כי הקבצים libmtm.a, set.h ו-list.h שסופקו לכם ימצאו בתיקיה הראשית ולקמפל לפי ההנחיות שבסוף התרגיל. שימו לב כי libmtm.a ו-libas.a קומפלו על השרת למחשב לינוקס, ולכן יכולים לא לעבוד במחשבים אחרים. תריצו אותם על השרת.
- לכל פונקציה המחזירה MatamikyaResult מפורטים ערכי השגיאה האפשריים לה. במקרה שפונקציה מקבלת קלט שמתאימים לו יותר מערך שגיאה אחד, יש להחזיר את ערך השגיאה הראשון כפי שמופיע בתיעוד הפונקציה בקובץ ה-h.
- עליכם להניח כי לא יתכנו מקרי שגיאה פרט לאלו המצוינים בתיעוד של כל פונקציה. בניגוד לאמור לעיל עבור שגיאה המעידה על בעיית זיכרון יש להחזיר מכל פונקציה בה היא מתרחשת את הערך MATAMIKYA_OUT_OF_MEMORY.
  במידה והתרחשה שגיאה על המערכת להישמר כאילו לא התבצעה הפעולה הגרמה לשגיאה.
  אתם יכולים להוסיף עוד קבצים במימוש שלכם, כל הקבצים שאתם מוסיפים יהיו בעלי שמות המתחילים ב-"matamikya".

1.3 דרישות נוספות לחלק הרטוב

1.3.1 Makefile

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

- הכלל הראשון ב-Makefile יקרא matamikya ויבנה את התוכנית matamikya.
- הקובץ יכיל כלל בשם amount_set_str שיבנה תוכנית המריצה כמה טסטים על המימוש של Amount Set.
- אנו מצפים לראות **שלכל ADT קיים כלל אשר בונה עבורו קובץ o.** דבר שכפי שלמדתם בקורס – אמור לחסוך הידור של כל התוכנית כאשר משנים רק חלק קטן ממנה.
- הוסיפו גם כלל clean, אשר מוחק את כל תוצרי הקמפול (מחזיר את סביבת העבודה למצב "נקי").
- יש לכתוב את הקובץ כפי שנלמד וללא שכפולי קוד.

תוכלו לבדוק את ה-makefile שלכם באמצעות הרצת הפקודות "make" או "make amount_set_str" והפעלת קבצי ההרצה שנוצרו. הנכם רשאים להשתמש בקבצים בתיקית tests על מנת לספק ל-Makefile פונקצית main.

1.3.2 הידור, קישור ובדיקה

התרגיל ייבדק על שרת csl3 ועליו לעבור הידור בעזרת הפקודות הבאות:

- עבור Amount Set:

```
gcc -std=c99 -Wall -pedantic-errors -Werror -DNDEBUG -o amount_set_str
amount_set_str*.c
```

- עבור מתמיקאה:

```
gcc -std=c99 -Wall -Werror -pedantic-errors -DNDEBUG -o matamikya
matamikya*.c tests/matamikya*.c -L. -lmtm -lm -las
```

עליכם לוודא שהרצה של פקודות אלו על csl3 אכן יוצרת את התוכניות הנדרשות מכם.

1.3.3 ולגרינד ודליפות זיכרון

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. על כן עליכם להשתמש ב-valgrind שמתחקה אחר ריצת התוכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך לבדוק האם יש לכם דליפות בתוכנית 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 ./matamikya`
   כאשר matamikya זה שם קובץ ההרצה.

הפלט ש-valgrind מפיק אמור לתת לכם, במידה שיש לכם דליפות, את שרשרת הקריאות שהתבצעו שגרמו לדליפה. אתם אמורים באמצעות דיבוג להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התוכנית. בנוסף, valgrind מראה דברים נוספים כמו קריאה לא חוקית (למשל קריאה לזיכרון שכבר שוחרר) – גם שגיאות אלו עליכם להבין מהיכן מגיעות ולתקן.

1.3.4 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 התוכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח בבדיקה שכזו, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות.

2 אופן ההגשה

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 מופיעים בתיקיית השורש בתוך קובץ ה-zip. הפתרון של החלק היבש יהיה בקובץ pdf שנקרא "dry.pdf", קבצים הרלוונטיים רק ל-Amount Set יהיו בעלי שמות המתחילים ב-"amount_set_str" ואותו דבר לגבי matamikya, הקבצים הרלוונטיים יתחילו ב- "matamikya". אין להגיש תיקייה שנקראת tests בתוך ה- zip.
- **יש להגיש אך ורק את קבצי ה-h וה-c אשר כתבתם בעצמכם ואת ה-makefile אשר נדרשתם לכתוב. אין להגיש את הקבצים אשר סופקו לכם.**
- הקבצים אשר מסופקים לכם יצורפו על ידינו במהלך הבדיקה. בפרט, ניתן להניח את קיום הקבצים list.h, set.h, amount_set.h, matamikya.h, libas.a ו-libmtm.a בתיקיה הראשית, ו-test_utilities.h בתיקייה tests.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.

- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית, שמרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף. כמו כן שמרו עותק של התרגיל של חשבון ה-csl3 שלכם לפני ההגשה האלקטרונית ואל תשנו אותו לאחריה (שני הקובץ יגרור שינוי חתימת העדכון האחרון).
    - כל אמצעי אחר לא ייחשב הוכחה לקיום הקוד לפני ההגשה.

לנוחותכם, אנו מספקים סקריפט בשם finalcheck.py לשימושכם לצורך וידוא תקינות ההגשה. הסקריפט מוודא שה-zip מכיל רק את הקבצים הנדרשים, ומנסה להדר את הקוד ולהריץ אותו. להרצת הסקריפט, הריצו את השורה הבאה (כאשר ex1_sol.zip הוא ה-zip שאתם עומדים להגיש):

```
~mtm/public/2122a/ex1/finalcheck.py ex1_sol.zip
```

זכרו, הסקריפט הוא לצורכי נוחות בלבד, וזו עדיין אחריותכם לוודא שההגשה עומדת בכל התנאים.

בהצלחה!