תרגיל בית 3 – ADT

הקדמה

בתרגיל זה נעבור על עקרונות ה-ADT כפי שנלמדו בקורס. בחלק הראשון של התרגיל נממש מחסנית גנרית שיכולה להכיל כל טיפוס בשפה, בחלק השני נשתמש ב-ADT שיצרנו ובחלק השלישי נייצר קובץ makefile לכלל הקבצים שכתבנו. אנו נספק לכם קובץ main המכיל בדיקה של מבנה הנתונים, זכרו לכלול אותו בתהליך הקומפילציה! שימו לב לתכנן את הקוד כראוי ולהקפיד על הקונבנציות שנלמדו בקורס.

תזכורת

מחסנית הוא מבנה נתונים מופשט העובד בשיטת LIFO (כלומר Last In First Out). למחסנית יש מספר פעולות בסיסיות: אתחול, הכנסה ושליפה. יש פעולות נוספות שניתן להגדיר על מחסנית שנממש את חלקן בתרגיל. בתרגיל זה נממש מחסנית עם גודל מקסימלי, כל ניסיון הכנסה של איבר מעבר למספר המקסימלי מחזיר כישלון.

חלק ראשון – Stack

בחלק זה תממשו ADT של מחסנית בקבצים stack.h, stack.c. נתונות ההצהרות הבאות:

מצביע לאיבר ב-stack	`typedef void * elem_t;`
מצביע לפונקציה המקבלת מצביע לאיבר מייצרת עותק שלו ומחזירה אותו	`typedef elem_t (*clone_t)(elem_t e);`
מצביע לפונקציה המקבלת מצביע לאיבר ומשחררת את הזיכרון שלו	`typedef void (*destroy_t)(elem_t e);`
מצביע לפונקציה המקבלת מצביע לאיבר ומדפיסה אותו	`typedef void (*print_t)(elem_t e);`

על ה-ADT לתמוך בפעולות הבאות:

- stack_create
  - ערכיי קבלה: גודל מחסנית מקסימלי ו-"השלימו לבד"
  - ערך החזרה: מצביע ל-ADT של מחסנית ריקה חדשה, NULL במקרה של כישלון
  - פעולה: מייצרת מחסנית ריקה חדשה
- stack_destroy
  - ערכיי קבלה: מצביע ל-ADT של מחסנית שיש למחוק
  - ערך החזרה: הצלחה\כישלון לפי קונבנציות הקורס
  - פעולה: הורסת את המחסנית כך שלא ייווצרו זליגות
- stack_push

- ערכיי קבלה : מצביע ל-ADT ומצביע לאיבר חדש
- ערך החזרה : הצלחה\כישלון לפי קונבנציות הקורס
- פעולה : מוסיפה עותק של האיבר לראש המחסנית

- stack_pop
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : אין
  - פעולה : מוציא את האיבר האחרון שהוכנס למחסנית ומוחק אותו
- stack_peek
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : מצביע לאיבר, NULL במקרה של כישלון
  - פעולה : מחזיר מצביע לאיבר האחרון שהוכנס למחסנית, NULL עבור כישלון
- stack_size
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : size_t, 0 במקרה של כישלון
  - פעולה : מחזיר את מספר האיברים הקיימים כרגע במחסנית
- stack_is_empty
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : bool, false במקרה של כישלון
  - פעולה : מחזיר true אם המחסנית ריקה, אחרת false
- stack_capacity
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : size_t, 0 במקרה של כישלון
  - פעולה : מחזיר את מספר המקומות הפנויים במחסנית, קרי גודל מקסימלי פחות מספר האיברים שיש כרגע במחסנית
- stack_print
  - ערכיי קבלה : מצביע ל-ADT
  - ערך החזרה : אין
  - פעולה : מדפיס את איברי המחסנית לפי סדר הכנסתם מהסוף להתחלה, כלומר האיבר האחרון שהוכנס יהיה הראשון שיודפס.

חלק שני – Student

בחלק זה נשתמש במחסנית שיצרנו בקבצים student.h, student.c. נגדיר אובייקט חדש בשפה מסוג סטודנט, להלן רשומה מסוג סטודנט:

```
struct student {
     char *name;
     int age;
     int id;
};
```

עליכם לממש את פונקציות המשתמש המועברות ל-ADT של המחסנית:

- student_clone – פונקציה המקבלת מצביע לאיבר מסוג סטודנט, יוצרת עותק מלא שלו ומחזירה את המצביע לעותק שנוצר. במקרה של כישלון מחזירה NULL.
- student_destroy – פונקציה המקבלת מצביע לאיבר מסוג סטודנט, ומשחררת את הזיכרון כולו.
- student_print – פונקציה המקבלת מצביע לאיבר מסוג סטודנט ומדפיסה אותו באופן הבא:

```
student name: STUDENT_NAME, age: AGE, id: ID.\n
```

שימו לב בסוף ההדפסה לבצע ירידת שורה (\n) מסומן באדום. שימו לב שהפונקציות האלה מתאימות בטיפוס ההחזרה ובפרמטרים להגדרות שהוגדרו בסעיף א'.

חלק שלישי – קמפול ויצירת makefile

עליכם ליצור קובץ makefile כנלמד בקורס אשר מקפל את כלל הקבצים בתכנית לקובץ בר הרצה.

תהליך הקומפילציה בקורס מבוא למערכות תוכנה 044101:

Makefile name: makefile (lower case)

Compiler: gcc

Compilation flags: -g -Wall -std=c99

Linker: gcc

Linker flags: -o

Make clean:

rm -rf *.o *.exe

בדיקת קובץ **make**:

עליכם לוודא כי קובץ ה- makefile עומד בקריטריונים הבאים:

- ביצוע קומפילציה מלאה ע"י קריאה ל- make.
- עדכון אחד מקבצי התוכנית אינו גורר פקודות קומפילציה מיותרות.

- ביצוע make clean למחיקת קבצי objects וקבצי executable.
- רשימת התלויות תכיל target rule יחיד לכל קובץ .c בתוכנית.
- שם קובץ ההרצה צריך להיות – prog.exe

מבנה קובץ ה- **makefile** : מבנה הקובץ מתחלק לשני חלקים :

<u>חלק א' :</u> מכיל הגדרות של משתנים, לדוגמא : הגדרת שם הקומפיילר, הגדרת שם ה- linker. עליכם להגדיר את המשתנים הבאים :

1. CC
2. CFLAGS
3. CCLINK
4. OBJS
5. EXEC – executable file name
6. RM

ערכי המשתנים נקבעים בהתאם להגדרות שניתנו בתחילת התרגיל. שימו לב, עליכם להשתמש בכל המשתנים לצורך הגדרת חלק ג' של קובץ ה- makefile. יש להשתמש בכל משתנה לפחות פעם אחת.

<u>חלק ב' :</u> רשימת תלויות, פקודות קומפילציה, פקודות linking ופקודת המחיקה לביצוע ניקיון של תיקיית ההרצה. עליכם לרשום את רשימת החוקים המגדירים את התלויות בין קבצי התוכנית. ודאו כי רשימת התלויות נכונה ואינה כוללת תלויות מיותרות. תחילה עליכם לרשום את פקודת הקישור, לאחר מכן את פקודות הקומפילציה ולבסוף את פקודת הניקיון. לדוגמא :

```
$(EXEC) : $(OBJS)
     $(CCLINK) …

a.o : a.c a.h b.h
     $(CC) …

b.o : b.c b.h
     $(CC) …

c.o : c.c c.h b.h
     $(CC) …
clean:
     …
```

זכרו כי כל שורת פקודה (השורה שבאה לאחר הכלל) חייבת להתחיל ב- **tab**. אין להתחיל או לסיים שורה ברווח.

חלק רביעי – בדיקת תקינות הקוד

השתמשו בקובץ main שסיפקנו לכם יחד עם ה-makefile שכתבתם ושאר קבציי הקוד שמימשתם על מנת לבדוק את תקינות הקוד שלכם. סיפקנו לכם קובץ קלט ופלט בסיסיים לבדיקת הקוד שלכם, מומלץ לכתוב טסטים נוספים לבדיקת מקריי קצה. הפקודות נקראות מה-STDIN מפוענחות ומפעילות את הפונקציות המתאימות. הפקודות בהן הקובץ תומך:

1. `Create_stack <MAX_NUM_OF_ELEM> <STACK_ID>`
2. `Add_student <ID> <NAME> <AGE> <STACK_ID>`
3. `Remove_student <STACK_ID>`
4. `Peek_stack <STACK_ID>`
5. `Stack_size <STACK_ID>`
6. `Stack_is_empty <STACK_ID>`
7. `Stack_capacity <STACK_ID>`
8. `Print_stack <STACK_ID>`

קובץ ה-main יוצר 3 מחסניות ומבצע עליהן את כלל הפעולות מעל, שימו לב שבסוף כלל המחסניות נמחקות.
יש לוודא כי כלל הזיכרון שוחרר.
הטסט שאנו מספקים לכם בודק תקינות בסיסית של הקוד שכתבת, הבדיקה היא מול האובייקט סטודנט.
על מנת להבטיח כי אכן הקוד שלכם כתוב בצורה נכונה ומכסה את כלל המקרים מומלץ אך לא חובה לבצע את שני השלבים הבאים:

1. כתיבת טסטים נוספים בעזרת כלי LLMs כגון ChatGPT. אלו הם כלי עזר אך לא תמיד הם מדויקים לכן יש להתייחס אליהם ככלי הכוונה ולא כדרך לביצוע משימות. בקשו מהם לכתוב לכם טסטים למבנה הנתונים שכתבנו וממשו אותם בעזרת פונקציית ה-main שנתנו לכם, השימוש דורש קצת הסתגלות אך יכול לעזור לכם רבות בתרגיל הבית, בהמשך התואר ובהמשך בעבודה או באקדמיה. נסו לרשום משהו בסגנון: "write me a test for a stack ADT in c"
2. טסט שכולל אובייקט אחר – אנו נבחן את הגנריות של המבנה נתונים שלכם על ידי הכנסה של טיפוסים אחרים חוץ מ-student. כתבו טיפוס חדש, ממשו את הנדרש על מנת להפעיל את מבנה הנתונים ובחנו האם ההתנהגות של הקוד תואמת את מה שאתם מצפים. מומלץ לשתף את הטסטים ולבחון קבציי פלט של חבריכם לראות שאתם מקבלים התנהגות דומה.

שימו לב **לא** לשתף את הקוד שכתבתם אלא רק את הטסטים שכתבתם – חל איסור על העתקה ושיתוף העבודה שלכם להגשה עם סטודנטים אחרים, העתקות יטופלו בחומרה! שימו לב **לא** להגיש אף קובץ למעט מה שנדרש ממכם.

<u>הוראות הגשה:</u>

1. עברו היטב על הוראות ההגשה של תרגילי הבית המופיעים באתר טרם ההגשה! ודאו כי התכנית שלכם עומדת בדרישות הבאות:
   א. התכנית קריאה וברורה.
   ב. התכנית מתועדת היטב לפי דרישות התיעוד המופיעות באתר.
2. שאלות בנוגע לתרגיל יש להפנות לפורום השאלות במודל.
3. סיכום מפרט התרגיל:

סעיף	תיאור
נושא התרגיל	ADT
קבצי הקוד הנתונים	main.c common.h
קבצי הקלט והפלט הנתונים	test-1.in test-1.out
הקבצים שיש להגיש	stack.h stack.c student.h student.c makefile

בהצלחה!