הנחיות כלליות

יש לשלוח את הקבצים באמצעות מערכת ההגשה לפני חלוף התאריך 5/1/20.

ניתן להגיש את התרגיל באיחור עם קנס אוטומטי על פי הפירוט הבא:

- יום איחור - קנס של 5 נקודות (ציון מקסימלי – 95).
- יומיים איחור - קנס של 15 נקודות (ציון מקסימלי – 85).
- שלושה ימי איחור - קנס של 30 נקודות (ציון מקסימלי – 70).

לאחר מכן לא יהיה ניתן להגיש את התרגיל (ציון 0).

המתרגל האחראי על התרגיל הוא אייל.

שאלות בנוגע לתרגיל יש לפרסם **באופן ציבורי בפורום הקורס** בלבד! רק אם לא התקבלה תשובה לאחר 24 שעות, יש לשלוח מייל לכתובת eyal.dayan@biu.ac.il עם קישור לדיון הרלוונטי.

בקשות להארכה מסיבות מוצדקות (מילואים, לידה וכו') יש לפרסם **באופן פרטי בפורום הקורס** בלבד (יש למען את הפוסט ל-instructors). בכל בקשה יש לציין שם מלא, שם משתמש במערכת ההגשה, תעודת זהות והאם אתם ממדעי המחשב או מתמטיקה.

יש להקפיד מאוד על הוראות עיצוב הקלט והפלט, בדיוק על פי הדוגמאות המצורפות.
שימו לב להנחיות במסמך ה-Coding Style המפורסם באתר הקורס.

עליכם לכתוב קוד על פי ההנחיות ולוודא שקיבלתם 100 בבדיקה האוטומטית הראשונית, וכן שהתרגיל מתקמפל ורץ על שרתי המחלקה (u2) ללא שגיאות או אזהרות.
תרגיל שלא עומד בסטנדרטים הבסיסיים הללו יגרור **ירידה משמעותית בציון התרגיל**, בשל הטרחה שהוא מייצר בתהליך הבדיקה שלו.

להזכירכם העבודה היא אישית. "עבודה משותפת" דינה כהעתקה. התרגיל נבדק על ידי מערכת ההגשה האוטומטית גם מהבחינה הזו, ותרגיל שהועתק יגרור ציון 0 **לכל הגורמים** השותפים בהעתקה. אתם יכולים לדון בגישות לפתרון התרגיל באופן תיאורטי, אך אין לשתף קוד בשום צורה.

בפיתוח הקוד ניתן להשתמש בכל סביבת עבודה, העיקר הוא שתדעו איך לקחת את קבצי הקוד מתוך הסביבה הזו, לבדוק אותם על שרתי האוניברסיטה ולהגיש אותם באמצעות מערכת ההגשה.
דוגמאות לחלק מהסביבות האפשריות:

IDEs (Integrated Development Environment):

- Visual studio
- Clion
- Eclipse
- Xcode

Text Editors:

- Atom
- Sublime
- Notepad++
- Vim

בהצלחה!

תרגיל 5 – ass5

בתרגיל זה עליכם לממש פונקציות בשלושה קבצי C בשם : stack.c queue.c strings.c

משקל התרגיל מתוך ציון התרגול : 15%.

התרגיל מורכב מארבעה מודולים נפרדים (קובץ C + קובץ H) המקיימים ביניהם יחסי גומלין (קריאה לפונקציות).

- לתרגיל זה מצורפים קבצי ה-H הבאים : element.h stack.h queue.h strings.h
- בנוסף מצורפים קבצי ה-C הבאים : element.c main.c
  אין להגדיר פונקציית main בקבצים שאתם מגישים. בנוסף, אין צורך להגיש את הקבצים המצורפים.
- בתרגיל זה מותר להשתמש בספריות stdio.h, stdlib.h, string.h, ctype.h, assert.h.
- בתרגיל זה נעשה שימוש בהקצאה דינמית.
  עליכם לוודא שלא נותר זיכרון שאינו משוחרר (דליפת זיכרון) בסיום ההרצה.
  הנושא הזה נבדק אוטומטית ומדווח לכם בפידבק המיידי במייל.

בקובץ output.txt מצורף הפלט עבור הקוד המופיע בקובץ main.c (כאשר הפונקציות ממומשות).

בחלקו הראשון של התרגיל, עליכם לממש מחסנית גנרית.

בחלקו השני של התרגיל, עליכם להשתמש במחסנית שמימשתם בחלק הראשון על מנת לממש תור גנרי.

בחלקו השלישי של התרגיל, עליכם להשתמש במחסנית שמימשתם בחלק הראשון על מנת לפתור בעיה הקשורה למחרוזות.

מבוא – גנריות

על מנת שנוכל לכתוב את הקוד של המחסנית והתור באופן שיהיה מנותק מטיפוסי הנתונים שיוכנסו אל תוך המבנים האלו, נגדיר את המודול Element ונשתמש רק בו במהלך הגדרת הפונקציות שלהם.

הגדרת Element דורשת הגדרה של הטיפוס עצמו, ושל הדרך לקלוט אותו מהמשתמש ולהדפיס אותו למסך.

בדוגמאות שלנו נשתמש בטיפוס Element כדי לייצג תו בודד. המודול נתון לכם ואין צורך לממש אותו.

```
typedef struct {
   char c;
} Element;

void printElement(Element e);
void scanElement(Element* e);
```

```
void printElement(Element e) {
   printf("%c", e.c);
}
void scanElement(Element* e) {
   scanf(" %c", &e->c);
}
```

חלק ראשון – מחסנית

בקובץ stack.h מופיעה הגדרה עבור טיפוס המייצג מחסנית. בנוסף, מופיעות בקובץ הצהרות המגדירות את הפעולות הבסיסיות שנדרשות על מנת לממש מחסנית, ועוד כמה פעולות נוספות. עליכם לממש את הפונקציות האלו בקובץ stack.c (כל פונקציית עזר שתרצו לממש עבור הפונקציות האלו, תימצא גם כן בקובץ הזה).

הגדרת הטיפוס:

בתרגיל שלנו נממש את המחסנית בעזרת מערך. מכיוון שאין חסם על כמות האיברים שניתן להכניס למחסנית, עליכם להשתמש במערך המוקצה באופן דינמי.

מחסנית

- מצביע למערך מסוג Element.
- גודל המערך.
- האינדקס של האיבר הנמצא בראש המחסנית.

```
typedef struct {
    Element* content;
    int size;
    int topIndex;
} Stack;
```

על מנת לנהל את ההקצאות באופן יעיל (נרצה לא לתפוס הרבה מקום בלי סיבה, ומצד שני לא נרצה להקצות ולשחרר זיכרון עבור כל הוצאה או הכנסה למחסנית) יש לדאוג לכך שגודל המערך יהיה תמיד חזקה של 2.

כאשר נגלה שאין לנו מספיק זיכרון, נכפיל את גודל ההקצאה.
כאשר נגלה שאנו משתמשים ביותר מדי זיכרון, נשחרר חצי מהגודל המוקצה.

עליכם לנהל את המחסנית כך שעבור מספר כלשהו של איברים, הגודל של המערך יהיה תמיד אותו הדבר, בלי קשר למצב שבו המחסנית הייתה קודם לכן.

כדי לדאוג לכך נגדיר את המצב הראשוני של המחסנית להיות מערך שגודלו 1, והאינדקס של האיבר בראש המחסנית יהיה 1- (מחסנית ריקה). בנוסף, נגדיר שתמיד צריך להיות במחסנית מקום פנוי (לאחר סיום ביצוע פעולה כלשהי).

הגדרת הפעולות:

אתחול (initStack) – הפונקציה מקצה זיכרון עבור מחסנית ריקה ומחזירה מצביע אליה.

שחרור (destroyStack) – הפונקציה משחררת את כל הזיכרון שהוקצה עבור המחסנית.

האם המחסנית ריקה (isStackEmpty) – הפונקציה מחזירה 1 אם המחסנית ריקה, 0 אם לא.

כמות האיברים במחסנית (lenOfStack) – הפונקציה מחזירה את מספר האיברים הנמצאים בתוך המחסנית.

קיבולת המחסנית (capacityOfStack) – הפונקציה מחזירה את גודלו של המערך המוקצה כרגע בזיכרון.

דחיפה (push) – הפונקציה מקבלת Element ומכניסה אותו לראש המחסנית.

שליפה (pop) – הפונקציה מחזירה את האיבר הנמצא כרגע בראש המחסנית, ומוציאה אותו מהמחסנית.

הצצה (top) – הפונקציה מחזירה את האיבר הנמצא כרגע בראש המחסנית, מבלי לשנות את המחסנית.

הדפסה (printStack) – הפונקציה מדפיסה את כל תוכן המחסנית (דוגמה תופיע בקובץ output.txt).

ניתן להניח שהמשתמש יבדוק בעזרת הפונקציה isStackEmpty שהמחסנית לא ריקה, לפני שינסה להפעיל את הפונקציה pop/top. מומלץ להשתמש ב-assert עבור מקרה שבו הפונקציה נקראת כך בכל זאת.

בכל מקרה של כישלון בהקצאת זיכרון יש להדפיס הודעת שגיאה בתוספת שם הקובץ ושם הפונקציה, כדי לקבל מידע אינפורמטיבי על הבעיה. בפונקציה initStack יש להחזיר NULL. בפונקציות push/pop אין לשנות את המחסנית (מעבר להכנסת / הוצאת Element).

חלק שני – תור

בקובץ queue.h מופיעה הגדרה עבור טיפוס המייצג תור. בנוסף, מופיעות בקובץ הצהרות המגדירות את הפעולות הבסיסיות שנדרשות על מנת לממש תור, ועוד כמה פעולות נוספות. עליכם לממש את הפונקציות האלו בקובץ queue.c (כל פונקציית עזר שתרצו לממש עבור הפונקציות האלו, תימצא גם כן בקובץ הזה).

הגדרת הטיפוס:

בתרגיל שלנו נממש את התור בעזרת שתי מחסניות. מכיוון שאין חסם על כמות האיברים שניתן להכניס לתור, ניתן להשתמש במחסנית שמימשתם בחלק הראשון.

תור

- מצביע למחסנית אחת.
- מצביע למחסנית שניה.

```
typedef struct {
   Stack* s1;
   Stack* s2;
} Queue;
```

האלגוריתם למימוש התור

- הכנסת איבר לתור תתבצע על ידי הכנסתו למחסנית s1.
- הוצאת איבר מהתור תתבצע באופן הבא:
  - אם s2 ריקה, יש להעביר את s1 לתוך s2 (לפי הסדר – הראשון שיוצא מs1 ייכנס ראשון לs2).
  - יש להוציא את האיבר הנמצא בראש המחסנית s2 ולהחזיר אותו.

הגדרת הפעולות:

אתחול (initQueue) – הפונקציה מקצה זיכרון עבור תור ריק ומחזירה מצביע אליו.

שחרור (destroyQueue) – הפונקציה משחררת את כל הזיכרון שהוקצה עבור התור.

האם התור ריק (isQueueEmpty) – הפונקציה מחזירה 1 אם התור ריק, 0 אם לא.

כמות האיברים בתור (lenOfQueue) – הפונקציה מחזירה את מספר האיברים הנמצאים בתוך התור.

קיבולת התור (capacityOfQueue) – הפונקציה מחזירה את גודלם של המערכים המוקצים כרגע בזיכרון.

הכנסה (enqueue) – הפונקציה מקבלת Element ומכניסה אותו לראש התור.

הוצאה (dequeue) – הפונקציה מחזירה את האיבר הנמצא כרגע בראש התור, ומוציאה אותו מהתור.

הצצה (peek) – הפונקציה מחזירה את האיבר הנמצא כרגע בראש התור, מבלי לשנות את התור.

ניתן להניח שהמשתמש יבדוק בעזרת הפונקציה isQueueEmpty שהתור לא ריק, לפני שינסה להפעיל את הפונקציה dequeue/peek. מומלץ להשתמש ב-assert עבור מקרה שבו הפונקציה נקראת כך בכל זאת.

בכל מקרה של כישלון בהקצאת זיכרון יש להדפיס הודעת שגיאה בתוספת שם הקובץ ושם הפונקציה, כדי לקבל מידע אינפורמטיבי על הבעיה. בפונקציה initQueue יש להחזיר NULL. בפונקציות enqueue/dequeue אין לשנות את התור (מעבר להכנסת / הוצאת Element).

חלק שלישי

בקובץ strings.h מופיעה חתימת הפונקציה הבאה:

```
int isLegalString(char str[]);
```

הגדרת הבעיה:

הפונקציה מקבלת מחרוזת המכילה סוגי סוגריים שונים:

רגילים: (), מרובעים: [], מסולסלים: { }, משולשים: <>.

הפונקציה בודקת האם המחרוזת "תקינה" – כלומר האם לכל סוגר שמאלי יש סוגר ימני מתאים, וביניהם יש גם כן מחרוזת "תקינה".

- שימו לב שלמרות שניתן לפתור את הבעיה בעזרת רקורסיה, אנחנו נפתור אותה בעזרת מחסנית.

עליכם לבדוק בעזרת המחסנית שמימשתם בחלק הראשון את המחרוזת, ולהחזיר 1 אם היא תקינה, 0 אם לא.

מספר דוגמאות למחרוזות

מחרוזת	isLegalString
123	1
(123)	1
{12[3]}	1
{123]	0
<<123>	0
123)	0

האלגוריתם לפתרון הבעיה

- נעבור על המחרוזת משמאל לימין.
- כל סוגר שמאלי יוכנס למחסנית.
- עבור כל סוגר ימני נבדוק את ראש המחסנית.
  - אם יש שם סוגר שמאלי מתאים נוציא אותו ונמשיך בבדיקה.
  - אם אין שם סוגר כזה – המחרוזת אינה תקינה.
- אם בסיום המעבר על המחרוזת המחסנית ריקה, המחרוזת תקינה.

הקובץ main.c

בקובץ הזה מופיעה פונקציית ה-main שמפעילה פונקציות המתפעלות את המודולים השונים.

הקובץ מיועד לבדיקות אישיות שלכם, וייתכן שיעודכן מדי פעם.

makefile

ניתן לקצר את קימפול הקבצים על השרת בעזרת קובץ makefile.

ניתן להגדיר בקובץ מערכת של תלויות, כך שרק הרכיבים הנחוצים יקומפלו.

הקובץ הזה נתון לכם יחד עם כל קבצי התרגיל, אתם לא נדרשים לכתוב קובץ כזה כרגע, אך עליכם להבין איך הוא עובד, כדי שתוכלו לכתוב קובץ כזה בעתיד.

פקודת הקימפול בתרגיל הזה תהיה make (בנוכחות הקובץ הזה). הפקודה מחפשת את היעד הראשון בתוך הקובץ ומפעילה את פקודת הקימפול המתאימה, אם כל היעדים מעודכנים.

אם קיים יעד שאינו מעודכן, הפקודה תפעיל קודם את הפקודה המתאימה עבור היעד הזה, לפי אותה הלוגיקה.

```
make
gcc -c main.c
gcc -c stack.c
gcc -c queue.c
gcc -c strings.c
gcc -c element.c
gcc main.o element.o stack.o queue.o strings.o
make
make: 'a.out' is up to date.
```

אם כל היעדים מעודכנים, הפקודה לא תעשה דבר, כפי שרואים בדוגמת הקימפול כאן.

אם רק חלק מהקבצים עודכנו, רק החלקים הרלוונטיים אליהם יקומפלו מחדש (למשל, שינוי של הקובץ queue.c לא דורש קימפול מחדש של stack.c).

בקובץ הזה הוספתי שני יעדים נוספים:

```
a.out: main.o stack.o queue.o strings.o element.o
	gcc -std=c99 main.o element.o stack.o queue.o strings.o
main.o: main.c
	gcc -c -std=c99 main.c
element.o: element.c element.h
	gcc -c -std=c99 element.c
stack.o: stack.c stack.h
	gcc -c -std=c99 stack.c
queue.o: queue.c queue.h
	gcc -c -std=c99 queue.c
strings.o: strings.c strings.h
	gcc -c -std=c99 strings.c
clean:
	rm -f *.o a.out
run: a.out
	./a.out
```

היעד clean מאפשר למחוק קבצים ישנים, כדי להכריח קימפול מחודש בנוחות (הפקודה להפעלתו תהיה make clean).

היעד run מאפשר לקמפל את כל התוכנית (אם צריך) ולהריץ אותה מיד (הפקודה להפעלתו תהיה make run).