מתרגל ממונה על התרגיל: יאיר פלדמן, yairf11@cs.technion.ac.il

תאריך ושעת הגשה: 05/12/2017 בשעה 23:30

אופן ההגשה: בזוגות.

הנחיות:

- שאלות לגבי דרישות התרגיל יש להפנות באימייל לכתובת הנ"ל.
- תשובות לשאלות המרכזיות אשר ישאלו יתפרסמו בחוצץ ה FAQ באתר הקורס לטובת כלל הסטודנטים.
  שימו לב כי **תוכן ה FAQ הוא מחייב וחובה לקרוא אותו**, אם וכאשר הוא יתפרסם.
  לא יתקבלו דחיות או ערעורים עקב אי קריאת ה FAQ.
- לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד **בכל** דרישות הסיבוכיות התרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול
- לצורך פתרון התרגיל, אם תבחרו לממש מבנה כלשהו של עץ, עליכם לממש אך ורק עצים מסוג **Splay Tree**.
  לצורך התרגיל, הניחו שסיבוכיות זמן הריצה של כל אחת מהפעולות find, insert, delete על splay tree היא $O(\log n)$ **משוערך**. בפרט, אין להשתמש בעצי AVL או B+. על סיבוכיות משוערכת תלמדו בהמשך הקורס.
  כדאי לעיין במצגת הבאה על מנת לקבל הבנה טובה יותר של ההתנהגות הצפויה של splay tree.

הקדמה:

קומודוס, קיסר האימפריה הרומית ה-18, הידוע בחיבתו העזה לקרבות גלדיאטורים, מעוניין להקים מערכת מחשוב חדשה שתאפשר לו לעקוב אחרי מאמני הגלדיאטורים והגלדיאטורים אותם הם קנו. הקיסר, ששמע על כישורי המחשוב המעולים של הסטודנטים בקורס מבני נתונים, החליט לפנות אליכם לצורך מימוש המערכת. תפקיד המערכת יהיה להחזיק מידע על כל המאמנים השונים איתם שומרת מועצת הקולוסיאום על קשר והגלדיאטורים השונים שקנה כל מאמן, כאשר כל גלדיאטור מחזיק מידע על רמת הלחימה שלו.

שימו לב: כל גלדיאטור הוא ייחודי ושייך למאמן אחד בלבד, הגלדיאטור מוגדר ע"י מזהה ייחודי.

דרוש מבנה נתונים למימוש הפעולות הבאות:

void* Init()

מאתחל מבנה נתונים ריק.

פרמטרים: אין.

ערך החזרה: מצביע למבנה נתונים ריק או $NULL$ במקרה של כישלון.

סיבוכיות זמן: $O(1)$ במקרה הגרוע.

StatusType AddTrainer(void *DS, int trainerID)

הוספת מאמן חדש עם המזהה *trainerID*

פרמטרים: DS מצביע למבנה הנתונים.
trainerID מזהה המאמן שצריך להוסיף.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.
INVALID_INPUT אם DS==NULL או trainerID <=0
FAILURE אם trainerID קיים.
SUCCESS במקרה של הצלחה.

סיבוכיות זמן: $O(k)$ במקרה הגרוע, כאשר k הוא מספר המאמנים.

StatusType BuyGladiator(void *DS, int gladiatorID, int trainerID, int level)

הוספת גלדיאטור חדש למערכת עם מזהה *gladiatorID*, השייך למאמן בעל המזהה *trainerID* ונמצא ברמה *level*.

פרמטרים:	DS	מצביע למבנה הנתונים.
	gladiatorID	מזהה הגלדיאטור שיש להוסיף.
	trainerID	מזהה המאמן.
	Level	הדרגה התחילית של הגלדיאטור
ערך החזרה:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם DS==NULL, gladiatorID<=0, trainerID<=0 או level<=0
	FAILURE	אם קיים כבר גלדיאטור עם מזהה gladiatorID או שהמאמן עם המזהה trainerID לא קיים.
	SUCCESS	במקרה של הצלחה.
סיבוכיות:	$O(log(n)+k)$ משוערך, כאשר n הוא מספר הגלדיאטורים ו-k הוא מספר המאמנים.	

StatusType FreeGladiator(void *DS, int gladiatorID)

לחופש נולד, שחרור הגלדיאטור בעל המזהה *gladiatorID*

פרמטרים:	DS	מצביע למבנה הנתונים.
	gladiatorID	מזהה הגלדיאטור שצריך להסיר.
ערך החזרה:	INVALID_INPUT	אם DS==NULL או gladiatorID <=0
	FAILURE	אם אין גלדיאטור עם מזהה gladiatorID.
	SUCCESS	במקרה של הצלחה.
סיבוכיות:	$O(log(n)+k)$ משוערך, כאשר n הוא מספר הגלדיאטורים ו-k הוא מספר המאמנים.	

StatusType LevelUp(void *DS, int gladiatorID, int levelIncrease)

האימון משתלם - העלאת רמת הלחימה של הגלדיאטור בעל המזהה *gladiatorID* ב-*levelIncrease* רמות.

פרמטרים:	DS	מצביע למבנה הנתונים.
	gladiatorID	מזהה הגלדיאטור שיש לעדכן.
	levelIncrease	מספר הרמות שיש להוסיף לגלדיאטור.
ערך החזרה:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם DS==NULL, gladiatorID <=0 או levelIncrease <=0
	FAILURE	אם אין גלדיאטור עם מזהה gladiatorID.
	SUCCESS	במקרה של הצלחה.
סיבוכיות:	$O(log(n))$ משוערך, כאשר n הוא מספר הגלדיאטורים.	

StatusType GetTopGladiator(void *DS, int trainerID, int * gladiatorID)

יש להחזיר את מזהה הגלדיאטור בעל הרמה הגבוהה ביותר מבין אלו ששייכים ל-*trainerID*

- *אם trainerID<0 יש להחזיר את הגלדיאטור בעל הרמה הגבוהה ביותר בכל המערכת, כלומר בין כל המאמנים.*
- *אם לשני גלדיאטורים יש level זהה, הגלדיאטור הטוב ביותר מביניהם יהיה בעל ה-*gladiatorID *הקטן יותר.*
- *אם אין גלדיאטורים ל-trainerID (או במערכת כולה אם trainerID <0) יש להחזיר -1 ב-*gladiatorID *, שימו לב שמקרה זה נחשב כ-SUCCESS.*

<u>פרמטרים</u>:	DS	מצביע למבנה הנתונים.
	trainerID	מזהה המאמן שעבורו נרצה לקבל את המידע.
	gladiatorID	מצביע למשתנה שיעודכן למזהה הגלדיאטור המתאים.
<u>ערך החזרה</u>:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם DS==NULL, gladiatorID ==NULL או trainerID == 0.
	FAILURE	אם trainerID>0 ואין מאמן עם מזהה trainerID
	SUCCESS	במקרה של הצלחה, כלומר כל מצב אחר.
<u>סיבוכיות</u>:	אם trainerID < 0 אז $O(1)$ במקרה הגרוע.	
	אחרת, $O(k)$ במקרה הגרוע. כאשר k הוא מספר המאמנים.	

StatusType GetAllGladiatorsByLevel(void *DS, int trainerID, int **gladiators, int *numOfGladiator)

יש להחזיר את כל הגלדיאטורים של המאמן בעל המזהה *trainerID* ממוינים על סמך הרמה שלהם.

- *אם trainerID <0 יש להחזיר את הגלדיאטורים בכל המערכת ממוינים על סמך הרמה שלהם.*
- *הגלדיאטורים יוחזרו ממוינים לפי רמה בסדר **יורד**, אם לשני גלדיאטורים יש אותה רמה אז יש למיין אותם בסדר **עולה** לפי gladiatorID.*
- *אם אין גלדיאטורים ל-trainerID (או במערכת כולה אם trainerID <0) יש להחזיר NULL ב-gladiators ואפס ב-numOfGladiator , שימו לב שמקרה זה נחשב כ-SUCCESS.*

<u>פרמטרים</u>:	DS	מצביע למבנה הנתונים.
	trainerID	מזהה המאמן שעבורו נרצה לקבל את המידע.
	gladiators	מצביע למערך שיכיל את כל הגלדיאטורים של המאמן.
	numOfGladiator	מצביע למשתנה שיעודכן למספר הגלדיאטורים במערך.
<u>ערך החזרה</u>:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם אחד המצביעים שווה ל-NULL או trainerID == 0.
	FAILURE	אם trainerID>0 ואין מאמן עם מזהה trainerID
	SUCCESS	במקרה של הצלחה, כלומר כל מצב אחר.
<u>סיבוכיות</u>:	אם trainerID < 0 אז $O(n)$ במקרה הגרוע, כאשר n הוא מספר הגלדיאטורים במערכת.	
	אחרת, $O(n_{\text{trainerID}} + k)$ במקרה הגרוע, כאשר $n_{\text{trainerID}}$ הוא מספר הגלדיאטורים של המאמן בעל המזהה trainerID ו-k הוא מספר המאמנים.	

שימו לב שאתם מקצים את המערך בגודל המתאים, כמו כן אתם צריכים להקצות את המערך בעצמכם באמצעות malloc (כי הוא ישוחרר בקוד שניתן לכם באמצעות free).

StatusType UpgradeGladiator(void *DS, int gladiatorID, int upgradedID)

הגלדיאטור *gladiatorID* ניצח הרבה קרבות ולכן צבר פרסום וקיבל שם חדש, דבר שנחשב לכבוד גדול בקרב גלדיאטורים ולכן נחשב לשדרוג. יש לעדכן את המזהה שלו ל-upgradedID , תוך שמירה על רמת הלחימה המקורית.

פרמטרים:	DS	מצביע למבנה הנתונים.
	gladiatorID	מזהה הגלדיאטור שעבר התפתחות
	upgradedID	המזהה החדש של הגלדיאטור, לאחר השדרוג
ערך החזרה:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם DS==NULL, gladiatorID <=0, upgradedID <=0
	FAILURE	אם אין גלדיאטור עם מזהה gladiatorID, או שקיים גלדיאטור עם מזהה upgradedID, ובפרט אם upgradedID == gladiatorID.
	SUCCESS	במקרה של הצלחה.
סיבוכיות:	$O(log(n))$ משוערך, כאשר n הוא מספר הגלדיאטורים.	

StatusType UpdateLevels(void *DS, int stimulantCode, int stimulantFactor)

הקיסר קומודוס גילה כי ישנם סמים ממריצים מיוחדים אשר יכולים לגרום להכפלה של רמתם של חלק מהגלדיאטורים. לכל סם מותאם *stimulantCode*, כך שבעת הפעלה הגלדיאטורים עבורם $gladiatorID \% stimulantCode == 0$ הם אלו שיושפעו. על הפונקציה, בהינתן *stimulantCode*, לעדכן, עבור כל גלדיאטור מתאים, את הרמה שלו ולהכפיל אותו ב-*stimulantFactor* .

פרמטרים:	DS	מצביע למבנה הנתונים.
	stimulantCode	הבסיס עליו פועל הסם.
	stimulantFactor	הפקטור שבו יש להכפיל את הרמה.
ערך החזרה:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם DS==NULL, stimulantCode <1 או stimulantFactor <1.
	SUCCESS	במקרה של הצלחה.
סיבוכיות:	$O(n+k)$ במקרה הגרוע. כאשר n הוא מספר הגלדיאטורים ו-k הוא מספר המאמנים.	

void Quit(void **DS)

הפעולה משחררת את המבנה. בסוף השחרור יש להציב ערך NULL ב-DS, אף פעולה לא תקרא לאחר מכן

פרמטרים:	DS	מצביע למבנה הנתונים.
ערך החזרה:	אין.	
סיבוכיות:	$O(n+k)$ במקרה הגרוע, כאשר n הוא מספר הגלדיאטורים ו-k הוא מספר המאמנים.	

סיבוכיות מקום - $O(n+k)$ במקרה הגרוע, כאשר n הוא מספר הגלדיאטורים ו-k הוא מספר המאמנים.

ערכי החזרה של הפונקציות:

בכל אחת מהפונקציות, ערך ההחזרה שיוחזר ייקבע לפי הכלל הבא:

- תחילה, יוחזר INVALID_INPUT אם הקלט אינו תקין.
- אם לא הוחזר INVALID_INPUT:

- בכל שלב בפונקציה, אם קרתה שגיאת הקצאה יש להחזיר ALLOCATION_ERROR.
- אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד FAILURE מבלי לשנות את מבנה הנתונים.
- אחרת יוחזר SUCCESS.

חלק יבש:

- הציון על החלק היבש הוא 50% מציון התרגיל.
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- יש להכין מסמך הכולל תיאור של מבני הנתונים והאלגוריתמים בהם השתמשתם בצירוף הוכחת סיבוכיות הזמן והמקום שלהם. חלק זה עומד בפני עצמו וצריך להיות מובן לקורא גם לפני העיון בקוד. אין צורך לתאר את הקוד ברמת המשתנים, הפונקציות והמחלקות, אלא ברמה העקרונית.
- ראשית הציגו את מבני הנתונים בהם השתמשתם. **רצוי ומומלץ להיעזר בציור**.
- לאחר מכן הסבירו כיצד מימשתם כל אחת מהפעולות הנדרשות. הוכיחו את דרישות סיבוכיות הזמן של כל פעולה תוך כדי התייחסות לשינויים שהפעולות גורמות במבני הנתונים.
- הוכיחו שמבנה הנתונים וכל הפעולות עומדים בדרישת סיבוכיות המקום.
- רמת פירוט: יש להסביר את כל הפרטים שאינם טריוויאליים ושחשובים לצורך מימוש הפעולות ועמידה בדרישות הסיבוכיות. אין לדון בפרטים טריוויאליים (הפעילו את שיקול דעתכם בקשר לזה, ושאלו את האחראי על התרגיל אם אינכם בטוחים). אין לצטט קטעים מהקוד כתחליף להסבר. אין צורך לפרט אלגוריתמים שנלמדו בכתה. כמו כן, אין צורך להוכיח תוצאות ידועות שנלמדו בכתה, אלא מספיק לציין בבירור לאיזו תוצאה אתם מתכוונים.
- הגשת החלק הרטוב מהווה תנאי הכרחי לקבלת ציון על החלק היבש, כלומר, הגשה בה יתקבל אך ורק חלק יבש תגרור ציון 0 על התרגיל כולו.
- על חלק זה לא לחרוג מ-8 עמודים.
- והכי חשוב **keep it simple!**

חלק רטוב:

- אנו ממליצים בחום על מימוש **Object Oriented**, ב**C++**, מימוש כזה יאפשר לכם להגיע לפתרון פשוט וקצר יותר לפונקציות אותן עליכם לממש ויאפשר לכם להכליל בקלות את מבני הנתונים שלכם (זכרו שיש תרגיל רטוב נוסף בהמשך הסמסטר). על מנת לעשות זאת הגדירו מחלקה, נאמר Colosseum, וממשו בה את דרישות התרגיל. אח"כ, על מנת לייצר התאמה לממשק ה C ב library1.h, ממשו את library1.cpp באופן הבא:

```
#include"library1.h"
#include"Colosseum.h"

void* Init() {
        Colosseum * DS = new Colosseum();
        return (void*)DS;
}
StatusType AddTrainer(void *DS, int trainerID){
        return ((Colosseum*)DS)-> AddTrainer (trainerID);
}
```

וכולי...

- על הקוד להתקמפל על *csl2* באופן הבא:
  - **g++ –DNDEBUG –Wall *.cpp**

עליכם מוטלת האחריות לוודא קומפילציה של התכנית ב+ + g. אם בחרתם לעבוד בקומפיילר אחר, מומלץ לקמפל ב+ + g מידי פעם במהלך העבודה.

הערות נוספות:

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ library1.h
- קראו היטב את הקובץ הנ"ל, לפני תחילת העבודה.
- אין לשנות את הקבצים הנ"ל ואין להגיש אותם.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל אין להשתמש במבנים של STL ואין להוריד מבני נתונים מהאינטרנט). **כחלק מתהליך הבדיקה אנו נבצע בדיקה ידנית של הקוד ונוודא שאכן מימשתם את מבני הנתונים שבהם השתמשתם.**
- יש לתעד את הקוד בצורה נאותה וסבירה.
- מסופקת לכם דוגמא של קובץ קלט (in.txt) וקובץ הפלט (out.txt) המתאים לו.
- שימו לב: התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט ארוכים, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.
- **יש לוודא שהפתרון שלכם אכן רץ ביעילות ביחס למימוש נאיבי** (שמממש רק מערכים, למשל). לצורך כך מסופקים לכם שלושה קבצים:
  - speed_exe – קובץ הרצה שמממש בצורה נאיבית את הפקודות: FreeGladiator, BuyGladiator, LevelUp. אין להניח שהפלט שלו נכון מבחינת דרישות התרגיל.
  - speed_test.in – קובץ קלט (ארוך) לדוגמא.
  - speed_run.sh – סקריפט bash שמריץ קובץ הרצה עם קובץ קלט מסוים (אותם הוא מקבל כקלט), ומדפיס את זמני הריצה של התוכניות – זו שקיבל כקלט וזו עם המימוש הנאיבי. לדוגמא, הדרך הנכונה להריץ את הסקריפט עם הקבצים המסופקים היא (הזמנים להמחשה בלבד):

```
>> chmod u+x speed_run.sh
>> ./speed_run.sh my_exe speed_test.in
Naive implementation time: 3.830
Your time: 0.912
Speedup is: 4.1995
```

שימו לב, זמן הריצה עשוי להשתנות בין מחשב למחשב, מה שחשוב הוא היחס בין זמני הריצה של התוכניות על אותו המחשב (speedup). לכן, נדרוש שהמימוש שלכם יהיה מהיר יותר מהמימוש הנאיבי המסופק, על הפעולות הבסיסיות הנ"ל. על מנת שהסקריפט ירוץ כראוי, הריצו אותו כך שהקובץ speed_exe נמצא באותה התיקייה שממנה אתם מריצים את הסקריפט. אין דרישה על ה speedup שתקבלו, אך פחות מ-1.5 צריך להדליק לכם נורת אזהרה לגבי המימוש שלכם.

הגשה:

- **חלק יבש+ חלק רטוב**:

הגשת התרגיל הנה **אך ורק** אלקטרונית דרך אתר הקורס.

יש להגיש קובץ **ZIP** (**ללא תיקיות או תתי תיקיות בתוכו**) שמכיל את הדברים הבאים:

- קבצי ה-Source Files שלכם (ללא הקבצים שפורסמו).
- קובץ PDF אשר מכיל את הפתרון היבש. מומלץ להקליד את החלק הזה. ניתן להגיש קובץ PDF מבוסס על סריקה של פתרון כתוב בכתב יד. שימו לב כי במקרה של כתב לא קריא, כל התרגיל לא ייבדק.
- קובץ submissions.txt, המכיל בשורה הראשונה את שם, תעודת הזהות וכתובת הדוא"ל של השותף הראשון ובשורה השנייה את שם, תעודת הזהות וכתובת הדוא"ל של השותף השני. לדוגמה:

Yair Feldman 012345678 yairf11@cs.technion.ac.il
Lucius Commodus 123456789 commodus@cs.technion.ac.il

- **שימו לב כי אתם מגישים את כל שלושת החלקים הנ"ל.**
- אין להשתמש בפורמט כיווץ אחר, מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים.
- אין להגיש קובץ המכיל תתי תיקיות.
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב.
- ההגשה האחרונה היא הנחשבת.

- הגשה שלא תעמוד בקריטריונים הנ"ל תפסל ותקנס בנקודות!

העתקות בתוכנה תטופלנה בחומרה! (Buh dum tss)

דחיות ואיחורים בהגשה:

- דחיות בתרגיל הבית תינתנה אך ורק לפי תקנון הקורס.
- 5 נקודות יורדו על כל יום איחור בהגשה ללא אישור מראש. באפשרותכם להגיש תרגיל באיחור של עד 5 ימים ללא אישור. תרגיל שיוגש באיחור של יותר מ-5 ימים ללא אישור מראש יקבל 0.
- במקרה של איחור בהגשת התרגיל יש עדיין להגיש את התרגיל אלקטרונית דרך אתר הקורס.
- במקרה של איחור מוצדק, יש לצרף לקובץ ה PDF שלכם את סיבות ההגשה באיחור, לפי הטופס המופיע באתר, כולל סריקות של אישורי מילואים או אישורי נבחן.

בהצלחה!