תרגיל בית תכנותי
להגשה עד 25.01.18 בשעה 23:50
בהצלחה!

תרגיל זה מנוסח בלשון זכר מטעמי נוחות בלבד והוא מיועד לכל המגדרים.
מתרגל אחראי על התרגיל: שמעון

<u>הוראות:</u>

1. יש להגיש את **כל** קבצי ה header וה cpp שיצרתם בקובץ **zip יחיד** בעל השם EXP_ID1_ID2 כאשר ID1 ו ID2 הם מספרי תעודות הזהות של שני בני הזוג. אין צורך להגיש קבצים שסופקו ע"י צוות הקורס.
2. ההגשה תתבצע רק ע"י **אחד** מבני הזוג.
3. **עליכם לוודא לפני הגשה כי הקוד שלכם מתקמפל ורץ בשרת ה t2** (הוראות מצורפות בקובץ נפרד).
4. זוג שהתרגיל שלו לא יתקמפל בשרת ה t2 או יעוף בזמן ריצה **ציונו בתרגיל יהיה 0.**
5. שים לב כי יש לשחרר כל זיכרון שהקצת. דליפת זיכרון תגרור הורדה בציון.
6. יש לכתוב קוד קריא ומסודר עם שמות משמעותיים למשתנים, למתודות ולמחלקות.

בתרגיל בית זה אתם מתבקשים לממש בשפת c++ מבנה נתונים דינמי המאפשר שמירה של מפתחות ומידע המשויך למפתח שיקרא BalancedTreeK. מבנה נתונים זה מהווה הכללה של עץ 2-3 לעץ מושרש בו מספר הילדים לכל צומת פנימי מלבד השורש הוא לפחות K ולכל היותר 2K-1. מספר הילדים של השורש הוא לפחות 2 ולכל היותר 2K-1.

למבנה הנתונים המתודות הפומביות הבאות:

- **BalancedTreeK(const Key* min ,const Key* max)** – בניית מבנה נתונים חדש עם פרמטר K, כאשר min הוא פוינטר לאובייקט מסוג Key עם המפתח המינימלי האפשרי ו max הוא פוינטר לאובייקט מסוג Key עם המפתח המקסימלי האפשרי. הפונקציה **מעתיקה** את המפתחות המוצבעים ע"י min ו max ומאתחלת את מבנה הנתונים. סיבוכיות נדרשת – $O(1)$.
- **~BalancedTreeK()** – destructor, הפונקציה מוחקת את מבנה הנתונים ואת כל המידע השמור בו. סיבוכיות נדרשת – $O(n)$.
- **void Insert(const Key* nkey, const Value* nval)** – הפונקציה **מעתיקה** את המפתח ש nkey מצביע עליו ואת המידע המשויך לו ש nval מצביע עליו ומוסיפה למבנה הנתונים את המפתח ואת המידע. סיבוכיות נדרשת – $O(\log n)$.
- **void Delete(const Key* dkey)** – הפונקציה מוחקת ממבנה הנתונים את האובייקט השמור אצלה בעל מפתח השווה למפתח ש dkey מצביע עליו ואת המידע המשויך אליו אם קיים מפתח כזה במבנה הנתונים, אחרת לא עושה כלום.
  סיבוכיות נדרשת - $O(\log n)$.

- **const Value* Search(const Key* key)** - הפונקציה מחזירה פוינטר למידע המשויך למפתח השווה למפתח ש key מצביע עליו אם קיים מפתח כזה במבנה הנתונים אחרת מחזירה NULL, ללא שינוי של מבנה הנתונים.
  סיבוכיות נדרשת - $O(\log n)$.
- **const unsigned Rank(const Key* key)** – אם קיים מפתח השווה למפתח ש key מצביע עליו במבנה הנתונים, הפונקציה מחזירה את המיקום של המפתח בסידור הלינארי (בסדר עולה) של המפתחות השמורים במבנה הנתונים אחרת הפונקציה מחזירה 0, ללא שינוי של מבנה הנתונים. סיבוכיות נדרשת - $O(\log n)$.
- **const const Key* Select(unsigned index)** – הפונקציה מחזירה מצביע למפתח במיקום index בסידור הלינארי (בסדר עולה) של המפתחות השמורים במבנה הנתונים. אם לא קיים מפתח במיקום זה הפונקציה מחזירה NULL, ללא שינוי של מבנה הנתונים. סיבוכיות נדרשת - $O(\log n)$.
- **const const Value* GetMaxValue(const Key* key1, const Key* key2)** – הפונקציה מחזירה מצביע למידע המקסימלי מבין המידע המשויך למפתחות בטווח [key1,key2] ללא שינוי של מבנה הנתונים. בכל מקרה אחר הפונקציה מחזירה NULL (שים לב כי ערך המפתחות key1 ו key2 לא בהכרח נמצאים במבנה הנתונים). סיבוכיות נדרשת $O(\log n)$.

עליכם לחשוב איך מחלקה תשתמש במחלקה אחרת, ואולי להגדיר מחלקות, משתנים ומתודות נוספות כרצונכם.

<u>הערה:</u>
שימו לב, כאשר עליכם להעתיק אובייקט מסוג Key או מסוג Value עליכם לעשות זאת באמצעות המתודה ()clone (הסבר על המתודה בהמשך) הקיימת באובייקטים Key ו Value בהתאמה.

<u>הסבר על הקבצים שקיבלתם:</u>

1. Key.h – חתימה של מחלקה אבסטרקטית (לא ניתן ליצור ממנה אובייקטים). מחלקה זו מספקת את הממשק עבור המפתחות שיוכנסו למבנה הנתונים. אינכם יודעים איך ימומש המפתח במבנה הנתונים (ב main שקיבלתם יש דוגמא), המפתח יכול להיות כל מחלקה שיורשת מ Key וממשת את המתודות הקיימות ב Key.
   על מנת ליצור אובייקט Key באפשרותכם להשתמש במתודה clone. מתודה זו מייצרת אובייקט חדש מסוג Key באמצעות אובייקט Key קיים ומחזירה פוינטר לאובייקט החדש. האובייקט החדש שנוצר הוא עותק של האובייקט עליו הפעלנו את הפונקציה clone. כמו כן, הזיכרון של האובייקט החדש מוקצה על ה heap (כלומר באופן דינמי).
   שים לב: לאחר שימוש במתודה clone נוצר אובייקט חדש (הפונקציה clone מבצעת את הפקודה new).
   לדוגמא, הפונקציה Insert יכולה לבצע:
   ```
   Key* key=nkey->clone();
   ```
   למשתנה key ייכנס מצביע לאובייקט חדש מסוג Key (שהוא עותק של nkey), כאשר הפונקציה clone שתקרא בזמן הריצה של התוכנית היא זו של המחלקה שיורשת מ Key.

2. Value.h – חתימה של מחלקה אבסטרקטית (לא ניתן ליצור ממנה אובייקטים).
מחלקה זו מספקת את הממשק עבור המידע שיוכנס למבנה הנתונים. אינכם יודעים איך ימומש המידע במבנה הנתונים (ב main שקיבלתם יש דוגמא) המידע יכול להיות כל מחלקה שיורשת מ Value וממשת את המתודות הקיימות ב Value.
על מנת ליצור אובייקט Value באפשרותכם להשתמש במתודה clone. מתודה זו מייצרת אובייקט חדש מסוג Value באמצעות אובייקט Value קיים ומחזירה פוינטר לאובייקט החדש. האובייקט החדש שנוצר הוא עותק של האובייקט עליו הפעלנו את הפונקציה clone. כמו כן, הזיכרון של האובייקט החדש מוקצה על ה heap (כלומר באופן דינמי).
שים לב: לאחר שימוש בפונקציה clone יש לכם אובייקט חדש (הפונקציה clone מבצעת את הפקודה new).
לדוגמא הפונקציה Insert יכולה לבצע:

```
Value* v=nval->clone();
```

למשתנה v ייכנס מצביע לאובייקט חדש מסוג Value (שהוא עותק של nval), כאשר הפונקציה clone שתקרא בזמן הריצה של התוכנית היא זו של המחלקה שיורשת מ Value.
3. BalancedTreeK.txt – קובץ טקסט ובו רשומות חתימות המתודות הפומביות שעליכם לממש במבנה הנתונים כפי שהוגדרו בתחילת התרגיל (מקובץ זה תוכל להעתיק את החתימות על מנת לוודא שיש לך את החתימה המדויקת הדרושה).
4. ParameterK.h – קובץ header המכיל את ההגדרה של הפרמטר K.
5. main.cpp – דוגמת הרצה ודוגמת מימוש של מפתח ושל מידע המשויך למפתח.
6. main_output – פלט לאחר הרצה של main.cpp בשרת ה t2.

<u>אילוצים:</u>

- הקוד שלך יכול להכיל include רק עבור מחלקות אותן אתה יצרת ו
#include < cstddef >.
על מנת לייצג מצביע ל NULL השתמש במילה השמורה NULL המוגדרת ב cstddef.
(בכל מקום שתצטרך להשתמש ב NULL הוסף #include < cstddef >).
כמו כן, תצטרך לעשות include במחלקות שיצרת ל Key.h, Value.h ו ParameterK.h.

<u>הדרכה:</u>

- אובייקטים מסוג Key תומכים אך ורק באופרטור > במידה ואתה נדרש, למשל, לאופרטור < האם באפשרותך לממשו בעזרת האופרטור >?
- אובייקטים מסוג Value תומכים אך ורק באופרטור > במידה ואתה נדרש, למשל, לאופרטור < האם באפשרותך לממשו בעזרת האופרטור >?
- אין להשתמש במתודות המוגדרות במחלקות MyKey ו MyValue ולא מוגדרות במחלקות Key ו Value אלו מימושים ספציפיים ולא בהכרח יהיו במימוש של המחלקות במהלך בדיקת התרגיל.

הנחות:

- הנח כי המפתחות שיוכנסו למבנה הנתונים יהיו ייחודיים, קטנים ממש מהערך המקסימלי שמאפשר Key וגדולים ממש מהערך המינימלי.
- הנח כי היחס > המוגדר על קבוצת ערכי המידע המשויכים למפתחות הוא יחס סדר מלא.
- בקריאה למתודות הפומביות של BalancedTreeK המקבלות כפרמטר מצביע ל Key מובטח כי המתודה תקרא עם ערך מפתח הקטן ממש מהערך המקסימלי האפשרי וגדול ממש מהערך המינימלי האפשרי.

הסבר על תהליך הבדיקה האוטומטית:

אנחנו נריץ את הקבצים שלכם עם פונקציית main משלנו (מצורפת דוגמא). שימו לב, כי פונקציית ה main שקיבלתם היא רק דוגמא, ייתכן ויוספו אובייקטים, סדר הפעולות ישתנה, גודל הקלט ישתנה, המימוש של המחלקות MyKey ו MyValue יהיו שונה, הפרמטר K ישתנה וכו'

במהלך הבדיקה יקומפלו **כל** הקבצים שהגשת (בתוספת Key.h, Value.h, ParameterK.h ו main בדיקה) בשרת ה t2 עם הפקודה

```
g++ *.cpp
```

לכן חשוב מאוד שתגישו את כל קבצי ה h וה cpp שיצרתם ותוודאו שהקוד מתקמפל בשרת.
בהנחה והקובץ מתקפל, הקוד יורץ והפלט של התוכנית יושווה לפלט חוקי.

המלצה חמה:

אל תשאירו את הבדיקה בשרת לרגע האחרון. ייתכן והקוד לא יתקמפל בשרת ותצטרכו לתקנו לפני ההגשה.