2. מבנה נתונים גנרי – SortedList

בחלק זה אסור להשתמש בSTL.

2.1. תיאור כללי

בחלק זה של התרגיל נממש מבנה נתונים גנרי ב++C תוך שימוש בתבניות, חריגות ואיטרטורים.

הערה: כיוון שבמועד פרסום התרגיל טרם נלמדו חריגות, מומלץ בתור התחלה לממש את המבנה ללא התחשבות בשגיאות, ולהוסיף שימוש בחריגות לאחר לימוד הנושא.

איטרטורים נלמדו בהרצאה 8, ומופי ם בשקפים 43-48. חזרה נוספת על איטרטורים תועבר בתרגול 10.

מבנה הנתונים אותו נממש הוא רשימה מקושרת ממוינת. והוא צריך לקיים את הדרישות הבאות:

1. כמות האיברים בSortedList אינה חסומה.
2. הוספת איברים מתבצעת ע"י פעולת insert
3. הסרת איברים מבוצעת ע"י פעולת remove
4. **איברי הרשימה ממוינים בכל רגע נתון –** פעולות insert וremove שומרות על הסדר בין איברי הרשימה. הסדר בין האיברים מוגדר ע"י אופרטור > של הטיפוס T.

לתבנית המחלקה SortedList יהיה פרמטר יחיד: class T. פרמטר זה יהיה טיפוס האלמנטים של הSortedList.

שימו לב: עליכם להניח כמות הנחות מינימלית על הטיפוס T. לא ניתן להניח שקיים בנאי חסר ארגומנטים לT.

2.2. ממשק הSortedList

המחלקה תספק את הממשק הבא:

1. בנאי חסר פרמטרים ליצירת SortedList ריק..
   הערה: זכרו להשתמש בעיקרון RAII (Resource Acquisition is Initialization) לתכנות נכון.
2. הורס לSortedList.
3. בנאי העתקה.
4. אופרטור השמה.
5. insert – מתודה שמקבלת אלמנט חדש כפרמטר ומכניסה אותו לרשימה.
6. remove – מתודה שמקבלת **איטרטור** (יפורט בהמשך) על המבנה כפרמטר ומסירה את האלמנט אליו הוא מפנה מהרשימה.

בהצלחה! ☺

7. length – מתודה שמחזירה את מס' האלמנטים ברשימה.
8. filter – מתודה שמקבלת **פרדיקט** (פונקציה בוליאנית) על טיפוס איברי הרשימה, ומחזירה רשימה חדשה שמכילה **רק את האיברים מהרשימה המקורית שמקיימים את התנאי**.
   הדרכה: יש להשתמש עבור מתודה זו בtemplate נוסף.
9. apply – מתודה שמקבלת **פונקציה** מטיפוס איברי הרשימה לטיפוס איברי הרשימה, ומחזירה SortedList שמכילה את תוצאות הפעלת הפונקציה על כל איברי הרשימה.
   הדרכה: יש להשתמש בטיפוס template נוסף.
10. begin – מתודה שמחזירה **איטרטור** לתחילת הרשימה.
11. end – מתודה שמחזירה **איטרטור** לסוף הרשימה.

2.3. איטרטור קבוע עבור SortedList

בנוסף לממשק הקיים, נממש מחלקת איטרטור קבוע עבור הרשימה, אשר תוגדר בתור SortedList::const_iterator.

על האיטרטור לספק את הממשק הבא:

1. בנאי העתקה ואופרטור השמה – מקבלים כפרמטר איטרטור אחר ומבצעים פעולה סטנדרטית.
   אסור לאפשר למשתמש גישה לבנאי הסטנדרטי של המחלקה.
2. הורס לאיטרטור.
3. ++operator – מימוש לאופרטור ++ שמקדם את האיטרטור לאיבר הבא ברשימה.
   אם האיטרטור מצביע לסוף הרשימה, **יש לזרוק חריגה** מטיפוס std::out_of_range.
4. ==operator – מימוש לאופרטור == שמקבל איטרטור נוסף ובודק האם שניהם שווים.
5. *operator - מימוש לאופרטור * שיחזיר **const reference** לאיבר אליו האיטרטור מפנה.

דגשים למימוש:

- אנחנו ממליצים להתחיל ממימוש של SortedList כרשימה **לא** גנרית ללא filter, apply.
  לאחר מכן לממש גנריות, ולבסוף לממש את filter וapply.
- את המימוש עליכם לכתוב בקובץ SortedList.h.
- יש לממש את המבנה כך שיהיה **לא תלוי** במשתמש מבחינת זיכרון. כלומר, עליו ליצור עותקים משלו לכל האובייקטים שהוא מקבל מהמשתמש.
- אתם רשאים להוסיף מתודות פרטיות ומחלקות עזר כרצונכם, תוך שמירה על עקרונות התכנות הנכון. אין לשנות את ממשק הרשימה או האיטרטור.
- הרשימה יכולה להכיל ערכים שווים מבחינת operator<. הדרישה היחידה היא סידור לפי הסדר שמגדיר האופרטור.

בהצלחה! ☺

דוגמת קוד להמחשת הממשק:

```
using std::string;
string getLen(string str)
{
    return std::to_string(str.length());
}

template<class T>
void printList(SortedList<T> list) {
    for (auto it = list.begin(); !(it == list.end()); ++it) {
        cout << *it << endl;
    }
    cout << endl;
}

int main()
{
    SortedList<string> lst1 = SortedList<string>();
    lst1.insert("Charlie");
    lst1.insert("Bob");
    lst1.insert("Alice");
    lst1.insert("Donald");
    printList(lst1);

    SortedList<string> lst2 = lst1;
    lst2 = lst2.apply(getLen);
    printList(lst2);

    SortedList<string>::const_iterator it = lst2.begin();
    cout << *it << endl << endl;
    ++it;
    lst2.remove(it);
    printList(lst2);
    return 0;
```

הפלט המתקבל עבור דוגמה זו הוא:

```
Alice
Bob
Charlie
Donald

3
5
6
7

3

3
6
7
```

בהצלחה! ☺