שאלות יבשות על SortedList

1. במבנה הSortedList הגנרי, מה הדרישות ההכרחיות שעל הטיפוס T לקיים? הסבירו.
תשובה:
על הטיפוס T לקיים את הדרישות ההכריחיות האלה:
1) operator> : אנחנו צריכים את האופרטור קטן ממש כדי שנשמור על סדר עולה ברשימה שלנו.
2) copy constructor : אנחנו צריכים את בנאי ההעתקה כדי שנוכל:
א. להעתיק את האיברים מטיפוס T מרשימת המקור, לרשימה חדשה, בבנאי ההעתקה של sorted list.
ב. להעתיק את האיבר שמקבלים כפרמטר בפונקציה insert אל תוך הרשימה.
3) destructor: כדי שנוכל לשחרר את הזכרון כשאנחנו הורסים את הרשימה, או מוחקים איבר מהרשימה באמצעות פונקצית remove.

2. נניח כי היינו רוצים לממש איטרטור non-const עבור SortedList. כלומר, עבור איטרטור זה האופרטור * היה מחזיר T&. איזו בעיה עלולה להיווצר במימוש?
תשובה:
אם היינו מממשים non-const iterator שמחזיר T&, אז המשתמש היה יכול לשנות בתוכן של האלמנטים מטיפוס T, בלי להשתמש בממשק שממשנו, דבר שיכול להרוס לנו את הסדר בין האיברים, וגם שובר לנו את ה encapsulation.

3. במתודה filter, המימוש כולל שימוש בtemplate נוסף כדי להעביר את הפרדיקט. אילו שתי דרכים שונות קיימות בשפת C++ למימוש והעברת הפרדיקט? מה ההבדל ביניהן? מדוע אנחנו לא צריכים לספק שני מימושים שונים כדי לתמוך בשתי הדרכים הללו?
תשובה:

שתי דרכים שונות שאפשר להשתמש בהם להעביר את הפרדיקט הן:

א. Function objects (על ידי שימוש בtemplate נוסף)
ב. std::function (מקרה פרטי של function objects)

אין הבדל מבחינת הsyntax של הקריאה לפונקציות אלו (קוראים לשתי הפונקציות כמו שקוראים לפונקציה רגילה למשל func(x)), זה קורה כיוון ששניהם עובדים באותה צורה,

כלומר, יצירת class עם overloaded function call operator()

הבדל אחד ביניהם הוא שאם משתמשים ב template נוסף, אנחנו לא מציינים מה הם ה Return and parameter types, אבל ב std::function כן מציינים אותם:

```
template<class Predicate> // First way: function objects

SortedList filter(Predicate c) const;

#include<functional> // Second way: std::function

SortedList filter(std::function<bool(const T&)>) const;
```