ספריית התבניות התקנית - איטרטורים

איטרטורים הם מושג מרכזי ב-STL. כל האלגוריתמים (שנלמד עליהם בהמשך) עובדים על איטרטורים. לדוגמה, אלגוריתם של חיפוש לא מבצע חיפוש על וקטור נתון, אלא חיפוש על תחום בין שני איטרטורים נתונים. לכן כדי להבין את הספריה חשוב להבין את סוגי האיטרטורים המוגדרים בה:

- איטרטור טריביאלי - משמש לקריאה בלבד, אי אפשר להזיז אותו.
- איטרטור קלט - משמש לקריאה בלבד, אפשר להזיז אותו קדימה (++).
- איטרטור פלט - משמש לכתיבה בלבד, אפשר להזיז אותו קדימה (++).
- איטרטור קדימה (`forward iterator`) - שילוב של איטרטור קלט ופלט, יכול לשמש לקריאה ולכתיבה.
- איטרטור דו-כיווני (`bidirectional iterator`) - כמו הקודם, רק שהוא יכול גם לזוז אחורה (--).
- איטרטור גישה אקראית (`random access iterator`) - כמו הקודם, רק שאפשר גם לבצע עליו אריתמטיקה כמו עם פוינטרים של סי.

המיכלים השונים מציעים איטרטורים ברמות שונות, למשל:

- **זרמים** מציעים איטרטורי קלט, פלט ואיטרטור "קדימה". למה צריך אותם? ראו דוגמה בתיקיה 1.
- **קבוצה, מפה ורשימה** מציעות איטרטור דו-כיווני.
- **וקטור** מציע איטרטור גישה אקראית.

לכל מיכל יש שיטה `begin` המחזירה איטרטור לתחילת המיכל ושיטה `end` המחזירה איטרטור ל**אחרי** סוף המיכל.

למיכלים המאפשרים הליכה אחורה (כמעט כולם, חוץ מזרמים ו-`forward_list`) יש גם שיטה `rbegin` המחזירה איטרטור לסוף המיכל ושיטה `rend` המחזירה איטרטור ל**לפני** תחילת המיכל (עבור איטרציה בסדר הפוך).

לכל איטרטור יש גם גירסה שהיא `const` - גירסה המאפשרת לקרוא את הפריטים במיכל אבל לא לשנות אותם. אפשר לגשת אליה ע"י `cbegin, cend, crbegin, crend`. איך מגדירים אותם? - ראו דוגמה בתיקיה 2.

יש גם איטרטורי הכנסה. למשל, לוקטור יש איטרטור בשם `back_insert_iterator`, שהוא איטרטור מסוג "פלט" (לכתיבה בלבד), ומשמש להכנסת פרטים חדשים בסוף הוקטור. לקבוצה יש איטרטור בשם `insert_iterator`, המשמש להכנסת פרטים לקבוצה (בהתאם לסדר המוגדר על הקבוצה). ראו דוגמה בתיקיה 1.

הכנסת פרטים בעזרת איטרטורים

כשרוצים להכניס פרט באמצע מיכל כלשהו (לא בסוף או בהתחלה), משתמשים בדרך-כלל באיטרטור שאומר איפה בדיוק להכניס.

- כדי להכניס פריט לפני המקום שהאיטרטור i מצביע עליו - **c.insert(i,x)**
- כדי להכניס פריטים בקטע החצי-פתוח מ-first ל-- last
  c.insert(i,first,last)
- אותו הדבר עם מחיקה - **erase**.
- אותו הדבר עם הכנסה במקום - **emplace**.

הפונקציות **insert, erase, emplace** עובדות בצורה דומה בכל המיכלים התומכים בהם (ראו טבלה ב **http://www.cplusplus.com/reference/stl**), אלא שהן שומרות על השמורה של המיכל. כך למשל, אם מנסים להכניס איבר למיכל מסודר, והאיטרטור שנותנים ל-**insert** מצביע למקום הלא נכון מבחינת הסדר - האיטרטור הזה ישמש רק כרמז (**hint**), החיפוש יתחיל משם אבל בסופו של דבר הפריט יוכנס למקום הנכון לפי הסדר.

תקינות איטרטורים

כשעובדים עם איטרטורים, חשוב לוודא שהם **תקינים**. מתי איטרטור עלול להיות לא-תקין? למשל, כשתוך כדי לולאה, אנחנו מוחקים את האיבר שהאיטרטור מצביע עליו:

- ברשימה, מפה וקבוצה - האיטרטור מכיל פוינטר המצביע למקום לא מאותחל בזיכרון - שגיאה חמורה.
- בוקטור - האיטרטור מצביע לאיבר הבא אחרי האיבר שמחקנו - לא שגיאה כל-כך חמורה, אבל עדיין לא מה שרצינו.

אז מה עושים? החל מ- C++11, השיטה erase מחזירה איטרטור מעודכן ותקין לאחרי המחיקה. צריך פשוט לשים את האיטרטור הזה באיטרטור שלנו. ראו דוגמה בתיקיה 3.

איטרטורים על מפה

כשמשתמשים באיטרטור i על מפה (map), הסוג של *i הוא זוג (pair) של מפתח+ערך. ראו תיקיה 4.

איטרטורים על מיכלים אסוציאטיביים

למיכלים אסוציאטיביים, כמו מפה או קבוצה, יש שיטות מיוחדות שמחזירות איטרטורים:

- find - מקבל מפתח, מחזיר איטרטור לערך המתאים או end() אם הערך לא נמצא.
- lower_bound - מקבל מפתח, מחזיר איטרטור לערך הכי קטן שהוא שווה או גדול מהמפתח.

- upper_bound - מקבל מפתח, מחזיר איטרטור לערך הכי קטן שהוא גדול ממש מהמפתח.

ראו תיקיה 4.

מקורות

- מצגות של אופיר פלא.
- Peter Gottschling, "Discovering Modern C++", chapter 4.
- תיעוד הספריה התקנית: http://www.cplusplus.com/reference/stl.

סיכם: אראל סגל-הלוי.