תבניות של מחלקות

[כ-60 דקות]

עד כאן ראינו תבניות של פונקציות. באותו אופן אפשר להגדיר תבניות של מחלקות.

נניח למשל שאנחנו רוצים להגדיר מחסנית. אפשר להגדיר מחסנית של מספרים, מחרוזות וכו'... הלוגיקה תהיה בדיוק אותו דבר.

לכן עדיף להגדיר את המחסנית כתבנית - `template`:

```
template <typename T> class Stack { ... }
```

שימו לב - כיוון שזו תבנית, צריך לממש את כל השיטות של המחלקה בקובץ h - אין לה קובץ cpp.

כדי להשתמש בתבנית הזאת, צריך להגיד לקומפיילר איזה סוג בדיוק לשים במקום T, למשל:

```
Stack<int> intList; // T = int

Stack<string> stringList; // T = string
```

ראו דוגמה בתיקיה 2.

איטרטורים

נניח שבנינו מחסנית כללית. עכשיו אנחנו רוצים להכניס לתוכה בבת-אחת מערך שלם של עצמים, למשל לכתוב: `stack.push(arr)`. איך נכתוב את השיטה push?

דרך אחת היא להעביר ל-push שני ארגומנטים: פוינטר למערך (arr), וכן את מספר האיברים במערך (כי הוא לא נמצא במערך -בניגוד לג'אבה).

דרך שניה, מקובלת מאד ב-C וגם ב-++C, היא להעביר שני ארגומנטים: פוינטר לתחילת המערך (arr) ופוינטר לסוף המערך (arr+length). זה מאפשר לבצע לולאה מהירה יותר הסורקת את כל המערך מהתחלה לסוף. בהמשך נראה שהשיטה הזו גמישה יותר.

אבל מה קורה אם המערך שממנו אנחנו מעתיקים הוא לא מערך פרימיטיבי של השפה, אלא מיכל כללי יותר, למשל, IntBuffer, LinkedList, ...? למיכל כזה כבר אין פוינטרים - ייתכן שהמידע בכלל לא נמצא בבלוק רציף בזיכרון!

האם אפשר לכתוב את הפונקציה push פעם אחת, כך שהיא תרוץ גם עם מערכים פרימיטיביים וגם עם מיכלים כלליים יותר?

התשובה היא כן. לשם כך צריך להגדיר **איטרטור**. איטרטור הוא עצם הצמוד למיכל כלשהו, ומתנהג כמו פוינטר, כלומר אפשר לכתוב בעזרתו קוד כמו:

```
for(; begin!=end; ++begin) { // Do something with *begin }
```

לשם כך צריך ליצור עצם שיש לו אופרטורים של פוינטר:

- השמה;
- הגדלה ב-1 (שני אופרטורים);

- שווה / לא שווה;
- אופרטור כוכבית (*) אונרי - גישה לעצם שהאיטרטור מצביע עליו.

במקרים מסויימים נרצה להוסיף אופרטורים נוספים:

- סוגריים מרובעים - גישה לאיברים שלא לפי הסדר;
- הקטנה ב-1 (חזרה אחורה);
- הגדלה / הקטנה במספר כלשהו.

איך יוצרים איטרטור? בדרך-כלל הוא יהיה מחלקה פנימית בתוך המחלקה שעליה הוא רץ. למשל בתוך המחלקה Stack תהיה מחלקה פנימית בשם iterator. אנחנו נוכל לגשת אליה באופן הבא:

```
Stack<int>::iterator i = myStack.begin();
```

עכשיו אנחנו יכולים להוסיף למחסנית שלנו שיטת push גנרית, או בנאי גנרי. הם יהיו template (כן, אפשר לכתוב שיטה שהיא תבנית בתוך מחלקה שהיא עצמה תבנית!). הפרמטר לתבנית יהיה סוג האיטרטור. כך, התבנית תוכל לקבל גם פוינטר רגיל של C, וגם איטרטור של מחסנית, או כל איטרטור אחר התומך באופרטורים של פוינטר.

ראו דוגמה בתיקיה 2.

יש כאן חיסכון גדול בכתיבת קוד. נניח שיש לנו m מבני-נתונים שונים. בלי תבניות, היינו צריכים m^2 בנאים - לבנות כל אחד מכל אחד אחר. עם תבניות, אנחנו צריכים רק m בנאים, ועוד איטרטור לכל מבני-נתונים, סה"כ כמות העבודה שלנו ירדה לבערך 2m.

לולאות עם איטרטורים

כשמחלקה מגדירה איטרטור ופונקציות begin, end, ניתן לרוץ על איברי המחלקה בלולאה באופן הבא:

```
for (Stk<int>::iterator i=mystack.begin(); i!=mystack.end(); ++i)
{
   int val = *i;
   cout << val << endl;
}
```

החל מ C++11, יש צורה קצרה יותר לכתוב את הלולאה הנ"ל:

```
for (int val: mystack) {
   cout << val << endl;
}
```

הפורמט הקצר נקרא foreach loop. הוא שקול לחלוטין לפורמט הארוך - הקומפיילר מתרגם את הפורמט הקצר לפורמט הארוך. לכן, כדי שהפורמט הקצר יעבוד, המחלקה צריכה להגדיר פונקציות begin ו-end המחזירות איטרטור עם אופרטורים מתאימים - ++, * וכו'.

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.
- Peter Gottschling, "Discovering Modern C++", chapter 3.

סיכם: אראל סגל-הלוי.