

העמסת אופרטורים - העתקה וחברים

בנאי מעתיק

דיברנו על סוגים מיוחדים של בנאים. אחד מהם הוא בנאי ללא פרמטרים. עוד בנאי מיוחד הוא **בנאי מעתיק**. בנאי מעתיק למחלקה T הוא בנאי המקבל פרמטר אחד בלבד, שהסוג שלו הוא:

`const T&`.

הקומפיילר קורא לבנאי הזה אוטומטית בכל פעם שצריך להעתיק עצם מהסוג T לעצם חדש, למשל, כשמעבירים פרמטרים לפונקציות.

אם לא מגדירים בנאי מעתיק, ברירת המחדל היא לבצע העתקת סיביות (`bitwise copy`). זה בסדר כשמדובר במחלקות פשוטות (`Point`) אבל לא טוב כשמדובר במבנים מורכבים עם מצביעים.

במקרה שברירת-המחדל לא מתאימה, אנחנו צריכים להגדיר בעצמנו בנאי מעתיק שיבצע "העתקה עמוקה".

חידה: מדוע הפרמטר חייב להיות מסוג `const T` ולא פשוט מסוג `T`?

תשובה: כי כדי להעביר פרמטר מסוג `T`, הקומפיילר צריך לקרוא לבנאי המעתיק, אבל אנחנו מגדירים את הבנאי הזה עכשיו!

ראו הדגמה של בנאי מעתיק בתיקיה 1.

אופרטור השמה

אופרטור השמה (=) מגדיר איך להעתיק עצמים. חשוב במיוחד להגדיר אותו נכון כשהעצמים הם "עמוקים" (כוללים הקצאת זיכרון דינמית). בדרך כלל, במקרה זה נרצה להגדיר שלוש שיטות: בנאי מעתיק (`copy constructor`), מפרק (`destructor`), ואופרטור השמה (`assignment operator`). ראו תיקיה 2.

אופרטורי המרה

סוג מיוחד של אופרטורים הוא **אופרטורי המרה**. הם הולכים ביחד עם **בנאי המרה**:

- בנאי המרה ממיר ממחלקה אחרת למחלקה שלנו. לדוגמה ראו במחלקה `Complex` בתיקיה 1, עבודה הגדרנו בנאי המרה ממספר ממשי.
- אופרטור המרה ממיר מהמחלקה שלנו למחלקה אחרת. לדוגמה ראו במחלקה `Fraction` בתיקיה 5, עבודה הגדרנו אופרטור המרה למספר ממשי.

האם אפשר לשים גם בנאי המרה וגם אופרטור המרה? -- אפשר, אבל זה עלול לבלבל את הקומפיילר, כי במקרים מסויימים יהיו לו שתי דרכים לבצע המרה והוא לא יידע באיזו מהן להשתמש. פתרון אפשרי הוא להשתמש במילה השמורה `explicit`. מילה זו אומרת לו להפעיל המרה רק כשמבקשים אותה באופן מפורש; ראו דוגמה בתיקיה 5.

המילה `explicit` נועדה גם למנוע שגיאות לוגיות ולהפוך אותן לשגיאות קומפילציה; ראו דוגמה בתיקה 6.

אופרטור גרשיים

אופרטור גרשיים נקרא גם אופרטור הסיומת (`suffix operator`). מגדירים אותו בעזרת גרשיים, אבל משתמשים בו כסיומת - בלי גרשיים. לדוגמה, מגדירים כך:

```
Complex operator"" _i(long double x) {
    return Complex(0,(double)x);
}
```

ומשתמשים פשוט כך:

```
7.0_i
```

ראו דוגמה בתיקה 8.

אופרטור פסיק

אופרטור פסיק מוגדר בשפת סי, והמשמעות שלו היא "התעלם מהפרמטר הראשון והחזר את הפרמטר האחרון". זה שימושי כשלפרמטר הראשון יש תוצאת-לוואי, למשל, הפרמטר הראשון מייצר ערך מסויים, והפרמטר השני בודק אותו.

בשפות-תיכנות מודרניות יותר, כגון פייתון, אופרטור פסיק משמש להגדרת `tuple` - זוגות, שלשות וכו'. אנחנו יכולים לקבל את המשמעות הזאת ע"י העמסה - אבל רק למחלקות או ל-`enum` (זה לא עובד עם מספרים).

ראו דוגמה בתיקה 9.

פונקציות ומחלקות חברות - friend

לפעמים יש פונקציה שהיא קשורה באופן הדוק למחלקה, אבל אי אפשר להגדיר בתוך המחלקה. לדוגמה, אופרטורי קלט ופלט (`<<` `>>`).

יש פונקציה שאפשר להגדיר בתוך המחלקה אבל נוח יותר להגדיר בחוץ. לדוגמה, אופרטור חיבור (+) אפשר להגדיר בתוך המחלקה:

```
Complex Complex::operator+ (Complex other)
```

אבל יותר טבעי להגדיר אותו מחוץ למחלקה:

```
Complex operator+ (Complex a, Complex b)
```

בדרך-כלל, כשפונקציות מוגדרות מחוץ למחלקה, אין להן גישה לשדות הפרטיים של המחלקה. אבל בשפת C++ אפשר לחרוג מכלל זה ע"י הצהרה שהפונקציה המדוברת היא **חברה** (`friend`) של

המחלקה. למשל, כדי להגדיר את אופרטור החיבור מחוץ למחלקה ולאפשר לו להשתמש בשדות של המחלקה, צריך לכתוב בתוך המחלקה (בקובץ h):

```
friend Complex operator+ (Complex a, Complex b);
```

את המימוש אפשר לכתוב כרגיל בקובץ cpp; המימוש יוכל לגשת לשדות הפרטיים של המחלקה.

כבר ראינו דוגמאות לזה במחלקה Complex; דוגמאות נוספות במחלקה Fraction בתיקיה 7.

באותו אופן, אפשר להגדיר **מחלקה חברה**: מחלקה אחת יכולה להגדיר מחלקה אחרת כ"חברה" ובכך לאפשר לה לגשת לשדות הפרטיים שלה.

זה שימושי כשיש שתי מחלקות הקשורות באופן הדוק זו לזו, למשל: מחלקה של וקטור, ומחלקה של איטרטור על הוקטור הזה.

חידה: האם הקשר הוא הדדי? אם מחלקה A חברה של מחלקה B, האם בהכרח מחלקה B גם חברה של מחלקה A? חישבו קודם מה התשובה ההגיונית, ואז כיתבו תוכנית דוגמה ובידקו אם זה אכן כך!

מקורות

- מצגת של אופיר פלא.
- <https://stackoverflow.com/q/49092801/827927>
- איך לכתוב אופרטור השמה עמוק בשתי שורות
<https://chara.cs.illinois.edu/sites/cgeigle/blog/2014/08/27/c/opy-and-swap>

סיכום: אראל סגל-הלוי.