העמסת אופרטורים

העמסה (overloading) היא מצב של כמה פונקציות עם אותו שם וארגומנטים מסוגים שונים.

בשפת ++C, גם אופרטור הוא פונקציה, ולכן אפשר **להעמיס אופרטורים**: להגדיר אופרטורים המבצעים פעולות שונות לפי סוג הארגומנטים המועברים אליהם. לדוגמה:

- **אופרטורים חשבוניים** (חיבור, חיסור, כפל, השוואה, וכו') מוגדרים על מספרים שלמים וממשיים; אנחנו יכולים להעמיס אותם גם במחלקות שאנחנו בונים, המייצגים עצמים מתמטיים מורכבים יותר. למשל: מספרים מרוכבים (ראו תיקיה 1), מטריצות, פולינומים וכו'. בספריה התקנית הם הורחבו גם למחרוזות.
- **אופרטורי זרימה** (<< >>) - מוגדרים במקור (בשפת סי) על מספרים שלמים, אבל בשפת ++C העמיסו אותם לזרמי קלט ופלט כפי שכבר ראינו. גם אנחנו יכולים להרחיב אותם כדי לכתוב ולקרוא מחלקות שאנחנו בונים (ראו תיקיה 1).
- **אופרטור סוגריים מרובעים []** - מוגדר לגבי מערכים בסיסיים; אפשר להרחיב אותו גם למחלקות שאנחנו בונים, כשאנחנו רוצים לגשת לדברים לפי אינדקס (ראו תיקיה 2). בספריה התקנית הם הורחבו למבנים כגון vector וגם map (ראו תיקיה 3).
- **אופרטור סוגריים עגולים ()** - יכול לשמש להגדרת אובייקטים המתפקדים כמו פונקציות - "פונקטורים" (functors). ראו תיקיה 4.

כשמעמיסים אופרטורים, חשוב לשים לב שהערך המוחזר תואם למשמעות של האופרטור. למשל:

- אופרטור + מחזיר את הסכום; אופרטור += מגדיל את הארגומנט השמאלי שלו אבל **גם** מחזיר את הסכום (שהוא הארגומנט השמאלי אחרי ההשמה).
- אופרטור = (השמה) מחזיר *this - זה מאפשר לבצע השמות בשרשרת.
- האופרטורים >> << מחזירים את זרם הקלט/פלט שהם מקבלים - שוב כדי לאפשר שרשרת.
- לאופרטור הגדלה באחד (++) והקטנה באחד (--) יש שתי גירסאות: כששמים אותו לפני המספר (prefix), הוא מחזיר את המספר אחרי ההגדלה; כששמים אותו אחרי המספר (postfix), הוא מחזיר את המספר לפני ההגדלה. כדי להבדיל בין האפשרויות, יש להגדיר את הגירסה השניה עם פרמטר מדומה מסוג int, למשל:
- `T& T::operator++();  // prefix operator`
- `T& T::operator++(int);  // postfix operator`

אופרטור השמה ובנאי מעתיק

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.

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 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, עבורה הגדרנו בנאי המרה ממספר ממשי.
- אופרטור המרה ממיר מהמחלקה שלנו למחלקה אחרת. לדוגמה ראו במחלקה Fraction בתיקיה 5, עבורה 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 בתיקיה 6.

אופרטור גרשיים

[המשך יבוא; תיקיה 8]

אופרטור פסיק

[המשך יבוא; תיקיה 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

סיכם: אראל סגל-הלוי.