

OO Programozás

Operátor felültöltés, másolás és átalakítás

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM119

2024. november 24.

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 839. 840. 84

Complex1.h

```
22     double getIm() const {
23         return im;
24     }
25     void setIm(double im) {
26         this->im = im;
27     }
28     void print() const {
29         std::cout << re << '+' << im << 'i' << std::endl;
30     }
31     Complex add(Complex right) const {
32         return Complex(re + right.re, im + right.im);
33     }
34     Complex mult(Complex right) const {
35         return Complex(re*right.re - im*right.im, im*right.re + re*right.im);
36     }
37 };
```

main1.cpp

```
1 #include <iostream>
2 #include "Complex1.h"
3
4 int main() {
5     Complex c(1., 1.);
6     Complex sum = c.add(Complex(2., 2.));
7     sum.print(); // 3+3i
8     Complex total = c.mult(Complex(3., 4.));
9     total.print(); // -1+7i
10 }
```

Operátor felültöltés

- A C++ megengedi, hogy az operátorok jelentését kiterjesszük a saját típusainkra (osztályainkra)
- Pl. ha értelmezhető az összeadás két int vagy float között, akkor két Complex objektum miért ne lehetne összeadható?
- Az operátorok működését (praktikusan nyilvános tag)függvények adják meg → `operatorX`, ahol X pl. +, *.

Complex2.h

```
31 // Use reference to avoid copy of 'right'
32 Complex operator+(const Complex& right) const {
33     return Complex(re + right.re, im + right.im);
34 }
35 Complex operator*(const Complex& right) const {
36     return Complex(re*right.re - im*right.im, im*right.re + re*right.im);
37 }
```

- + művelet bal operandusa: aktuális objektum, a jobb oldalt paraméterként kapja.
- Utóbbi felesleges (tagonkénti) másolásának elkerülésére referenciát használunk.
- Új, ideiglenes (eredmény) objektum jön létre e kettő alapján, ezt adja vissza.

main2.cpp

```
4  int main() {
5      Complex c(1., 1.);
6      // Complex sum = c.operator+(Complex(2., 2.));
7      Complex sum = c + Complex(2., 2.);
8      sum.print();
9      // Complex sum2 = c + 100.;
10     // error: no match for 'operator+'
11     // (operand types are 'Complex' and 'double')
12     // note: no known conversion for argument 1
13     // from 'double' to 'const Complex&'
14     Complex total = c * Complex(3., 4.);
15     total.print();
16 }
```


Probléma:

Összeadásnál a jobb oldali operandusnak Complex-nek kell lennie.

Megoldás:

További felültöltött operátor függvények hozzáadása, pl. double-t hozzáadhatunk a valós részhez.

Complex3.h

```
32     Complex operator+(double re) const {  
33         return Complex(this->re + re, im);  
34     }
```

main3.cpp

```
4  int main() {
5      Complex c(1., 1.);
6      // Complex sum = c.operator+(Complex(2., 2.));
7      Complex sum = c + Complex(2., 2.);
8      sum.print();
9      // Complex sum2 = c.operator+(100.);
10     Complex sum2 = c + 100.;
11     sum2.print();
12     // Complex sum3 = 100. + c;
13     // error: no match for 'operator+'
14     // (operand types are 'double' and 'Complex')
```

- Ez sem segít, ha a *bal* operandus `double` típusú → *nem tag*, két paraméteres operator függvény.
- Nem éri el a privát/védett tagokat → barát (`friend`) függvény: mindenhez hozzáfér az osztályon belül.
- Másik osztály is lehet az osztályunk barátja, pl.:
`friend class FriendOfComplex;`

Complex4.h

```
6  class Complex {
7      double re, im;
8      public:

28      friend std::ostream& operator<<(std::ostream& os, const Complex& cplx);
29
30      Complex operator+(const Complex& right) const {
31          return Complex(re + right.re, im + right.im);
32      }
33      Complex operator+(double re) const {
34          return Complex(this->re + re, im);
35      }
36      friend Complex operator+(double re, const Complex& right);
```

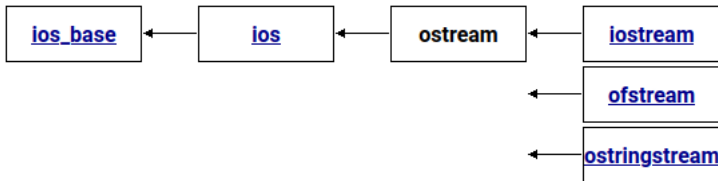
Complex4.cpp

```
3  std::ostream& operator<<(std::ostream& os, const Complex& c) {  
4      os << c.re << '+' << c.im << 'i';  
5      return os;  
6  }  
7  
8  Complex operator+(double re, const Complex& right) {  
9      return Complex(re + right.re, right.im);  
10 }
```

main4.cpp

```
4  int main() {
5      Complex c(1., 1.);
6      // Complex sum = c.operator+(Complex(2., 2.));
7      Complex sum = c + Complex(2., 2.);
8      std::cout << sum << std::endl; // 3+3i
9      // Complex sum2 = c.operator+(100.);
10     Complex sum2 = c + 100.;
11     std::cout << sum2 << std::endl; // 101+1i
12     // Complex sum3 = operator+(100., c);
13     Complex sum3 = 100. + c;
14     std::cout << sum3 << std::endl; // 101+1i
```

Az `ostream` a `cout`, `cerr` és `clog` objektumok típusa.



Bár különösebb haszna nincs, de csupa nem tag barát függvénnyel is megoldhattuk volna az operátorok felültöltését.

Complex5.h (Complex5.cpp)

```
6  class Complex {
7      double re, im;
8      public:

28      friend std::ostream& operator<<(std::ostream& os, const Complex& cplx);
29
30      friend Complex operator+(const Complex& left, const Complex& right);
31      friend Complex operator+(const Complex& left, double re);
32      friend Complex operator+(double re, const Complex& right);
33
34      friend Complex operator*(const Complex& left, const Complex& right);
35      friend Complex operator*(const Complex& left, double re);
36      friend Complex operator*(double re, const Complex& right);
37  };
```


Operátor felültöltés

- Definiáltuk az operátor *jelentését* olyan kifejezésben, melyben legalább egy operandus az osztály objektuma.
- Nem változtatható meg az operátor *szintakszisa*, *precedenciája*, *asszociativitása*, *operandusainak száma*.
- *Majdnem* minden operátor felültölthető.

- Az értékadás megengedett objektumok között → tagonkénti másolás.
- Probléma: dinamikusan foglalt területet több objektum adattagja is címezhet. Ha az egyik destruktora ezt felszabadítja, a másik ezen próbálhat műveletet végezni vagy újra felszabadítani.
- Egészítsük ki a **Message** osztályt felültöltött értékadás operátorral, készítsünk másolatot a tárolt szövegről!

message2.cpp

```
4  class Message {
5      private:
6          char* pStr;
7      public:
8          Message() {
9              pStr = new char('\0');
10             std::cout << "Created_[" << this << "]\n";
11         }
12
13         Message(const char* s) {
14             pStr = new char[strlen(s) + 1];
15             strcpy(pStr, s);
16             std::cout << "Created_[" << this << ",_" << pStr << "]\n";
17         }
```

message2.cpp

```
19 ~Message() {
20     std::cout << "Freed [" << this << ", " << pStr << "]\n";
21     delete[] pStr;
22 }
23
24 friend std::ostream& operator<<(std::ostream& os, const Message& m);
25
26 // return reference instead of void to allow multiple assignments
27 Message& operator=(const Message& m) {
28     // do not copy to itself
29     if(&m == this) return *this;
30     setMessage(m.pStr);
31     return *this;
32 }
```

message2.cpp

```
34     void setMessage(const char* s) {  
35         delete[] pStr;  
36         pStr = new char[strlen(s) + 1];  
37         strcpy(pStr, s);  
38     }  
39 };  
40  
41 std::ostream& operator<<(std::ostream& os, const Message& m) {  
42     os << m.pStr;  
43     return os;  
44 }
```

message2.cpp

```
46 int main() {  
47     Message m1;  
48     m1.setMessage(" Hello_C++_world!");  
49     std::cout << m1 << std::endl;  
50  
51     Message m2;  
52     m2 = m1;  
53     std::cout << m2 << std::endl;  
54  
55     Message m3;  
56     m3 = m2 = m1;  
57     std::cout << m3 << std::endl;  
58  
59     m1 = m1;  
60     std::cout << m1 << std::endl;  
61  
62     Message m4 = m1; // double free!  
63     std::cout << m4 << std::endl;  
64 }
```

Kimenet

```
Created [0x7ffcf5018618]  
Hello C++ world!  
Created [0x7ffcf5018620]  
Hello C++ world!  
Created [0x7ffcf5018628]  
Hello C++ world!  
Hello C++ world!  
Hello C++ world!  
Freed [0x7ffcf5018630, Hello C++ world!]  
Freed [0x7ffcf5018628, Hello C++ world!]  
Freed [0x7ffcf5018620, Hello C++ world!]  
Freed [0x7ffcf5018618, ]  
free(): double free detected in tcache 2  
Aborted (core dumped)
```

Ami jól működik:

Az értékadás operátor, ami másolatot hoz létre a karakterláncból,
nem engedi magát önmagára másolni, és
a többszörös értékadás is működik a visszaadott referencia miatt.

Ami hibás:

Az inicializáció nem értékadás, ilyenkor nem fut le a felültöltött függvényünk!

Másolási konstruktor (copy constructor):

Akár *egyetlen, azonos típusra vonatkozó referencia* paraméterrel is hívható konstruktor. (Lehetnek további paraméterek alapértelmezett értékekkel.)

Konverziós konstruktor (conversion constructor):

Ez is hívható egyetlen paraméterrel, de az más típusú.

Közös tulajdonság: objektum *inicializálásának* lehetősége kétféle szintaktikával:

```
Oszталy obj(42);   ≡   Oszталy obj = 42;
```


Másolási konstruktor további tulajdonságai:

- Ha mi nem készítünk, a fordító létrehoz egyet → tagonkénti másolás.
- Impliciten hívásra kerül:
 - Objektum átadása függvénynek paraméterként → a végtelen rekurzió elkerülése miatt kell az első paraméternek referenciának lennie.
 - Függvény adott osztály objektumát adja vissza.

message3.cpp

```
13 Message(const char* s) { // conversion ctor
14     pStr = new char[strlen(s) + 1];
15     strcpy(pStr, s);
16     std::cout << "Created " << this << ", " << pStr << "]\n";
17 }
18
19 /* for old compilers
20 Message(const Message& m) {
21     pStr = new char[strlen(m.pStr) + 1];
22     strcpy(pStr, m.pStr);
23     std::cout << "Created [" << this << ", " << m.pStr << "]\n";
24 } */
25
26 // C++11+ / delegating constructors
27 Message(const Message& m) : Message(m.pStr) {}
```

message3.cpp

```
56 int main() {  
57     Message m1;  
58     m1.setMessage("Hello_C++_world!");  
59     std::cout << m1 << std::endl;  
  
72     Message m4 = m1; // calls copy ctor  
73     std::cout << m4 << std::endl;  
74     Message m5(m1); // calls copy ctor  
75     std::cout << m5 << std::endl;  
76     Message m6 = "Conversion_+copy_ctor";  
77     Message m7("Conversion_ctor");  
78 }
```

Kimenet

```
Created [0x7fff0778e740]  
Hello C++ world!  
...  
Created [0x7fff0778e758, Hello C++ world!]  
Hello C++ world!  
Created [0x7fff0778e760, Hello C++ world!]  
Hello C++ world!  
Created [0x7fff0778e768, Conversion ctor]  
Created [0x7fff0778e770, Conversion ctor]  
Freed [0x7fff0778e770, Conversion ctor]  
Freed [0x7fff0778e768, Conversion ctor]  
Freed [0x7fff0778e760, Hello C++ world!]  
Freed [0x7fff0778e758, Hello C++ world!]  
...  
Freed [0x7fff0778e740, Hello C++ world!]
```

Mikor hívjuk a konverziós konstruktort?

- Explicit hívás, más típusú adattal.
- Implicit hívás, a fordító egy adatot az osztály típusára akar alakítani → lehetővé teszi a felültöltött operátorfüggvények számának csökkentését!

Complex6.h

```
6  class Complex {
7      double re, im;
8  public:
9      // explicitly defaulted constructor
10     Complex() = default;
11     // conversion constructor
12     Complex(double re, double im = 0.) {
13         this->re = re;
14         this->im = im;
15     }

29     friend Complex operator+(const Complex& left, const Complex& right);
30     friend Complex operator*(const Complex& left, const Complex& right);
31 };
```

Complex6.h

```
Complex() = default;
```

Ha készítünk legalább egy konstruktort, a fordító nem hoz létre automatikusan *alapértelmezett* konstruktort. Ha mégis szükség lenne rá: `default` (C++11+; használható a kulcsszó más *speciális* tagfüggvényekhez is.)

Legtöbbször ha szükség van az alábbiak közül akár csak egynek is a definiálására, akkor a másik kettőre is szükség van ([Rule of Three](#)):

- destruktork
- másoló konstruktor
- hozzárendelés operátor

Vagy, átmeneti jelleggel akár le is tilthatjuk egyes függvények alapértelmezett előállítását → delete

Hasznos lehet nem kívánt típuskonverziók tiltására is.

message4.cpp

```
4  class Message {  
  
7      public:  
  
19         // C++11: deleted constructor  
20         Message(const Message& m) = delete ;  
  
29         // deleted assignment operator  
30         Message& operator=(const Message& m) = delete ;  
  
37     };
```


message4.cpp

```
44  int main() {  
  
49      Message m2;  
50      // m2 = m1; error: use of deleted function  
51      // 'Message& Message::operator=(const Message&)'  
52  
53      // Message m4 = m1; error: use of deleted function  
54      // 'Message::Message(const Message&)'  
55  
56      // Message m6 = "Conversion + copy ctor"; error: use of deleted function  
57      // 'Message::Message(const Message&)'  
58      // after user-defined conversion: Message::Message(const char*)  
59  
60      Message m7("Conversion_ctor"); // OK  
61  }
```