Αντικειμενοστραφείς Γλώσσες Προγραμματισμού C++ / ROOT

Ιωάννης Παπαδόπουλος

Τμήμα Φυσικής, Πανεπιστήμιο Ιωαννίνων

Δεκέμβριος 2018

Περιεχόμενα

Περιεχόμενα

1. Ειδικές μέθοδοι κατασκευής αντικειμένων
   - Μέθοδοι κατασκευής αντιγράφου (copy constructor)
   - Κατασκευή μέσω τελεστή ανάθεσης (assignment operator)

2. Υπερφόρτωση τελεστών (operator overloading)
   - Παραδείγματα υπερφόρτωσης τελεστών
   - Παράδειγμα υπερφόρτωσης τελεστή <<
   - Παράδειγμα υπερφόρτωσης τελεστή +
   - Παράδειγμα υπερφόρτωσης τελεστή +=

Ειδικές μέθοδοι κατασκευής αντικειμένων

Ειδικές μέθοδοι κατασκευής αντικειμένων

Μέθοδοι κατασκευής αντιγράφου (copy constructors)

- Για μία κλάση, μία μέθοδος κατασκευής αντιγράφου είναι μία ειδική μέθοδος για την κατασκευή αντικειμένων μέσω αντιγραφής ενός ήδη υπάρχοντος στιγμιοτύπου της κλάσης.
- Η πιο συνηθισμένη υπογραφή μίας μεθόδου κατασκευής αντιγράφου είναι η ακόλουθη:

```
MyClass( const MyClass& other );
```

- Σε περίπτωση που δεν υλοποιηθεί από τον χρήστη μία μέθοδος κατασκευής αντιγράφου για μία κλάση, ο μεταγλωττιστής ορίζει μόνος του εμμέσως μία (η οποία φροντίζει απλώς για την αντιγραφή των αντίστοιχων μελών της κλάσης από το υπάρχον στιγμιότυπο στο αντίγραφο).

Ειδικές μέθοδοι κατασκευής αντικειμένων

Μέθοδοι κατασκευής αντιγράφου (copy constructors)

- Για μία κλάση, μία μέθοδος κατασκευής αντιγράφου είναι μία ειδική μέθοδος για την κατασκευή αντικειμένων μέσω αντιγραφής ενός ήδη υπάρχοντος στιγμιοτύπου της κλάσης.
- Η πιο συνηθισμένη υπογραφή μίας μεθόδου κατασκευής αντιγράφου είναι η ακόλουθη:

```
MyClass( const MyClass& other );
```

- Σε περίπτωση που δεν υλοποιηθεί από τον χρήστη μία μέθοδος κατασκευής αντιγράφου για μία κλάση, ο μεταγλωττιστής ορίζει μόνος του εμμέσως μία (η οποία φροντίζει απλώς για την αντιγραφή των αντίστοιχων μελών της κλάσης από το υπάρχον στιγμιότυπο στο αντίγραφο).

Κατασκευή μέσω ενός τελεστή ανάθεσης (assignment operator)

- Ένας τελεστής ανάθεσης (assignment operator) για μία κλάση επιτρέπει τη χρήση του τελεστή `'='` για την ανάθεση ενός προϋπάρχοντος στιγμιοτύπου σε ένα άλλο προϋπάρχον στιγμιότυπο της κλάσης.
- Υπάρχουν πολλές διαφορετικές υπογραφές για τον τελεστή ανάθεσης (operator =), π.χ.:

```
MyClass& operator=( const MyClass& other );
```

- Σε περίπτωση που δεν υλοποιηθεί από τον χρήστη κάποιος τελεστής ανάθεσης για μία κλάση, ο μεταγλωττιστής μπορεί (ανάλογα με τη δομή της κλάσης) να ορίσει εμμέσως μόνος του έναν (ο οποίος φροντίζει απλώς για την αντιγραφή των αντίστοιχων μελών της κλάσης από το στιγμιότυπο του δεξιού μέλους σε αυτό του αριστερού).

Μέθοδοι κατασκευής αντιγράφου (copy constructors)

- Γενικά, όταν τα αντικείμενα μίας κλάσης δεν περιλαμβάνουν μέλη που δεσμεύονται δυναμικά, δεν είναι απαραίτητο να υλοποιήσουμε μεθόδους κατασκευής αντιγράφου.

Μέθοδοι κατασκευής αντιγράφου (copy constructors)

- Γενικά, όταν τα αντικείμενα μίας κλάσης δεν περιλαμβάνουν μέλη που δεσμεύονται δυναμικά, δεν είναι απαραίτητο να υλοποιήσουμε μεθόδους κατασκευής αντιγράφου.
- Υπογραφές μεθόδων κατασκευής αντιγράφου:

```
MyClass(          const MyClass& other );      <-- προτιμώμενη
MyClass(                MyClass& other );
MyClass( volatile const MyClass& other );
MyClass( volatile       MyClass& other );
```

Μέθοδοι κατασκευής αντιγράφου (copy constructors)

- Γενικά, όταν τα αντικείμενα μίας κλάσης δεν περιλαμβάνουν μέλη που δεσμεύονται δυναμικά, δεν είναι απαραίτητο να υλοποιήσουμε μεθόδους κατασκευής αντιγράφου.
- Υπογραφές μεθόδων κατασκευής αντιγράφου:

```
MyClass(          const MyClass& other );      <-- προτιμώμενη
MyClass(                MyClass& other );
MyClass( volatile const MyClass& other );
MyClass( volatile       MyClass& other );
```

- Γιατί να χρησιμοποιήσω **const** ;...

```
1  // Προβληματική συνάρτηση, αν κληθεί με κάτι άλλο από μεταβλητή,
2  // αφού απαγορεύεται να χρησιμοποιηθεί αναφορά σε προσωρινό αντικείμενο.
3  void print_bad( std::string& s ) {   // όρισμα χωρίς const
4    std::cout << s << std::endl;
5  }
6
7  // Καλώς ορισμένη συνάρτηση:
8  void print_good( const std::string& s ) {
9    std::cout << s << std::endl;
10 }
11
12 std::string hello("Hello");
13
14 print_bad( hello );  // επιτυχής μεταγλώττιση, αφού το hello δεν είναι προσωρινό
15 print_bad( std::string("World") );  // λάθος, το όρισμα είναι προσωρινό
16 print_bad( "!" ); // λάθος, κατασκευάζεται προσωρινό std::string από const char*
17
18 print_good( hello );                // επιτυχής μεταγλώττιση
19 print_good( std::string("World") ); // επιτυχής μεταγλώττιση
20 print_good( "!" );                  // επιτυχής μεταγλώττιση
```

Κατασκευή μέσω τελεστή ανάθεσης (assignment operator)

- Πότε πρέπει να υλοποιήσουμε τελεστές ανάθεσης μια μία κλάση;
  Όταν πρέπει να υλοποιήσουμε και μεθόδους κατασκευής αντιγράφου (copy constructors).

Κατασκευή μέσω τελεστή ανάθεσης (assignment operator)

- Πότε πρέπει να υλοποιήσουμε τελεστές ανάθεσης μια μία κλάση;
  Όταν πρέπει να υλοποιήσουμε και μεθόδους κατασκευής αντιγράφου (copy constructors).
- Υπογραφές τελεστών ανάθεσης:

```
      MyClass& operator=( const MyClass& rhs );
      MyClass& operator=(       MyClass& rhs );      <-- προς αποφυγή
      MyClass& operator=(       MyClass  rhs );
const MyClass& operator=( const MyClass& rhs );
const MyClass& operator=(       MyClass& rhs );      <-- προς αποφυγή
const MyClass& operator=(       MyClass  rhs );
      MyClass  operator=( const MyClass& rhs );
      MyClass  operator=(       MyClass& rhs );      <-- προς αποφυγή
      MyClass  operator=(        MyClass rhs );
```

Κατασκευή μέσω τελεστή ανάθεσης (assignment operator)

- Πότε πρέπει να υλοποιήσουμε τελεστές ανάθεσης μια μία κλάση;
  Όταν πρέπει να υλοποιήσουμε και μεθόδους κατασκευής αντιγράφου (copy constructors).
- Υπογραφές τελεστών ανάθεσης:

```
      MyClass& operator=( const MyClass& rhs );
      MyClass& operator=(       MyClass& rhs );      <-- προς αποφυγή
      MyClass& operator=(       MyClass  rhs );
const MyClass& operator=( const MyClass& rhs );
const MyClass& operator=(       MyClass& rhs );      <-- προς αποφυγή
const MyClass& operator=(       MyClass  rhs );
      MyClass  operator=( const MyClass& rhs );
      MyClass  operator=(       MyClass& rhs );      <-- προς αποφυγή
      MyClass  operator=(        MyClass rhs );
```

- Παράδειγμα υλοποίησης ενός τελεστή ανάθεσης:

```
1  class MyClass {
2    int x;
3    char c;
4    std::string s;
5  };
6
7  MyClass& MyClass::operator=( const MyClass& other ) {
8    x = other.x;   // other : το αντικείμενο δεξιά του τελεστή =
9    c = other.c;
10   s = other.s;   // this  : δείκτης προς το τρέχον αντικείμενο,
11   return *this;  //         δηλαδή προς το αντικείμενο αριστερά του τελεστή =
12 }
```

Κατασκευή αντιγράφου VS Κατασκευή μέσω τελεστή ανάθεσης

```cpp
#include<iostream>
#include<stdio.h>

using namespace std;

class Test {
  public:
    Test() {}
    Test(const Test &t) {
      cout << "Copy constructor called" << endl;
    }
    Test& operator = (const Test &t) {
      cout << "Assignment operator called" << endl;
    }
};

int main() {
  Test t1, t2;   // Δημιουργία των στιγμιοτύπων t1 και t2
  t2 = t1;       // Στο t2 γίνεται ανάθεση του t1

  Test t3 = t1;  // Δημιουργία του στιγμιοτύπου t3 με αντιγραφή του t1 σε αυτό
                 // Είναι ισοδύναμο με:   Test t3(t1);
}

// έξοδος στην οθόνη:
// Assignment operator called
// Copy constructor called
```

Για περισσότερες πληροφορίες, κλικ εδώ: Copy constructor vs assignment operator in C++

Υπερφόρτωση τελεστών (operator overloading)

Υπερφόρτωση τελεστών (operator overloading) 1/2

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).

Υπερφόρτωση τελεστών (operator overloading) 1/2

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;

Υπερφόρτωση τελεστών (operator overloading)

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;
  1. Όταν το νόημα ενός τελεστή δεν είναι προφανές, κατανοητό ή αδιαμφισβήτητο, τότε δεν θα πρέπει να γίνει υπερφόρτωσή του. Στην περίπτωση αυτή είναι προτιμότερο να υλοποιήσει κανείς μία συνάρτηση επιλέγοντας ένα πολύ εύστοχο όνομα για αυτή.
     Εν ολίγοις, οι τελεστές δεν πρέπει να υπερφορτώνονται...

Υπερφόρτωση τελεστών (operator overloading)

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;
  1. Όταν το νόημα ενός τελεστή δεν είναι προφανές, κατανοητό ή αδιαμφισβήτητο, τότε δεν θα πρέπει να γίνει υπερφόρτωσή του. Στην περίπτωση αυτή είναι προτιμότερο να υλοποιήσει κανείς μία συνάρτηση επιλέγοντας ένα πολύ εύστοχο όνομα για αυτή.
     Εν ολίγοις, οι τελεστές δεν πρέπει να υπερφορτώνονται...
  2. Αν πρέπει να υπερφορτωθεί ένας τελεστής, θα **πρέπει να διατηρηθεί η καθιερωμένη του σημασία**. Για παράδειγμα, η υπερφόρτωση του τελεστή '+' θα πρέπει να σχετίζεται με την πρόσθεση και όχι με την αφαίρεση...

Υπερφόρτωση τελεστών (operator overloading)

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;
  1. Όταν το νόημα ενός τελεστή δεν είναι προφανές, κατανοητό ή αδιαμφισβήτητο, τότε δεν θα πρέπει να γίνει υπερφόρτωσή του. Στην περίπτωση αυτή είναι προτιμότερο να υλοποιήσει κανείς μία συνάρτηση επιλέγοντας ένα πολύ εύστοχο όνομα για αυτή.
     Εν ολίγοις, οι τελεστές δεν πρέπει να υπερφορτώνονται...
  2. Αν πρέπει να υπερφορτωθεί ένας τελεστής, θα **πρέπει να διατηρηθεί η καθιερωμένη του σημασία**. Για παράδειγμα, η υπερφόρτωση του τελεστή '+' θα πρέπει να σχετίζεται με την πρόσθεση και όχι με την αφαίρεση...
  3. Όταν υπερφορτωθεί ένας τελεστής, θα **πρέπει να υπερφορτωθούν και όλοι οι σχετιζόμενοι τελεστές**. Π.χ. αν έχει υπερφορτωθεί ο τελεστής '+' θα υπάρχει η παράσταση a+b, και αναμένει κανείς πως θα πρέπει να υπάρχει και η παράσταση a+=b (δηλαδή θα πρέπει να έχει υπερφορτωθεί και ο τελεστής '+=').

Υπερφόρτωση τελεστών (operator overloading)

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;
  1. Όταν το νόημα ενός τελεστή δεν είναι προφανές, κατανοητό ή αδιαμφισβήτητο, τότε δεν θα πρέπει να γίνει υπερφόρτωσή του. Στην περίπτωση αυτή είναι προτιμότερο να υλοποιήσει κανείς μία συνάρτηση επιλέγοντας ένα πολύ εύστοχο όνομα για αυτή.
     Εν ολίγοις, οι τελεστές δεν πρέπει να υπερφορτώνονται...
  2. Αν πρέπει να υπερφορτωθεί ένας τελεστής, θα **πρέπει να διατηρηθεί η καθιερωμένη του σημασία**. Για παράδειγμα, η υπερφόρτωση του τελεστή '+' θα πρέπει να σχετίζεται με την πρόσθεση και όχι με την αφαίρεση...
  3. Όταν υπερφορτωθεί ένας τελεστής, θα **πρέπει να υπερφορτωθούν και όλοι οι σχετιζόμενοι τελεστές**. Π.χ. αν έχει υπερφορτωθεί ο τελεστής '+' θα υπάρχει η παράσταση a+b, και αναμένει κανείς πως θα πρέπει να υπάρχει και η παράσταση a+=b (δηλαδή θα πρέπει να έχει υπερφορτωθεί και ο τελεστής '+=').
- Τελεστές που δεν μπορούν να υπερφορτωθούν:

```
.     ::     .*     ?:     sizeof     typeid
```

Υπερφόρτωση τελεστών (operator overloading)

- Οι περισσότεροι τελεστές της C++ μπορούν να επανοριστούν ή να υπερφορτωθούν, και να λειτουργούν ανάλογα με το τι είδους είναι οι τελεστέοι τους (δεξί ή/και αριστερό τους μέλος).
- Πότε θα πρέπει να γίνει υπερφόρτωση κάποιου τελεστή;
  1. Όταν το νόημα ενός τελεστή δεν είναι προφανές, κατανοητό ή αδιαμφισβήτητο, τότε δεν θα πρέπει να γίνει υπερφόρτωσή του. Στην περίπτωση αυτή είναι προτιμότερο να υλοποιήσει κανείς μία συνάρτηση επιλέγοντας ένα πολύ εύστοχο όνομα για αυτή.
     Εν ολίγοις, οι τελεστές δεν πρέπει να υπερφορτώνονται...
  2. Αν πρέπει να υπερφορτωθεί ένας τελεστής, θα **πρέπει να διατηρηθεί η καθιερωμένη του σημασία**. Για παράδειγμα, η υπερφόρτωση του τελεστή '+' θα πρέπει να σχετίζεται με την πρόσθεση και όχι με την αφαίρεση...
  3. Όταν υπερφορτωθεί ένας τελεστής, θα **πρέπει να υπερφορτωθούν και όλοι οι σχετιζόμενοι τελεστές**. Π.χ. αν έχει υπερφορτωθεί ο τελεστής '+' θα υπάρχει η παράσταση a+b, και αναμένει κανείς πως θα πρέπει να υπάρχει και η παράσταση a+=b (δηλαδή θα πρέπει να έχει υπερφορτωθεί και ο τελεστής '+=').
- Τελεστές που δεν μπορούν να υπερφορτωθούν:

  `.` `::` `.*` `?:` `sizeof` `typeid`

- Κατάλογος τελεστών που μπορούν να υπερφορτωθούν:
  - Αριθμητικοί τελεστές: `+ - * / % += -= *= /= %= ++ --`
  - Τελεστές για bits: `& | ^ << >> &= |= ^= <<= >>= ~`
  - Τελεστές Bool: `== != < > <= >= || && !`
  - Τελεστές μνήμης: `new new[] delete delete[]`
  - Διάφοροι: `[] -> ->* , * & ()`

Υπερφόρτωση τελεστών (operator overloading) 2/2

- Ένας δυαδικός τελεστής δέχεται ένα όρισμα αριστερά του και ένα όρισμα δεξιά του, π.χ. **a+b** . Μπορεί να οριστεί με δύο τρόπους:

.

Υπερφόρτωση τελεστών (operator overloading) 2/2

- Ένας δυαδικός τελεστής δέχεται ένα όρισμα αριστερά του και ένα όρισμα δεξιά του, π.χ. **a+b** . Μπορεί να οριστεί με δύο τρόπους:
  - Μέσω μίας μεθόδου που είναι μέλος της κλάσης, και δέχεται ως μοναδικό της όρισμα μία αναφορά σε αντικείμενο της κλάσης, π.χ.

```
MyClass& operator+(const MyClass& rhs);
```

.

Υπερφόρτωση τελεστών (operator overloading) 2/2

- Ένας δυαδικός τελεστής δέχεται ένα όρισμα αριστερά του και ένα όρισμα δεξιά του, π.χ. **a+b** . Μπορεί να οριστεί με δύο τρόπους:
  - Μέσω μίας μεθόδου που είναι μέλος της κλάσης, και δέχεται ως μοναδικό της όρισμα μία αναφορά σε αντικείμενο της κλάσης, π.χ.
    ```
    MyClass& operator+(const MyClass& rhs);
    ```
  - Μέσω μίας μεθόδου που δεν ανήκει στην κλάση και δέχεται δύο ορίσματα. Τουλάχιστον το δεύτερο όρισμα είναι μία αναφορά σε αντικείμενο της κλάσης. Για να μπορέσουν οι μέθοδοι αυτές να έχουν πρόσβαση στα ιδιωτικά δεδομένα της κλάσης, ορίζονται ως φίλες μέσω της λέξης friend, π.χ.
    ```
    friend std::ostream& operator<<(std::ostream& s, const MyClass& rhs);
    ```

.

Υπερφόρτωση τελεστών (operator overloading) 2/2

- Ένας δυαδικός τελεστής δέχεται ένα όρισμα αριστερά του και ένα όρισμα δεξιά του, π.χ. **a+b** . Μπορεί να οριστεί με δύο τρόπους:
  - Μέσω μίας μεθόδου που είναι μέλος της κλάσης, και δέχεται ως μοναδικό της όρισμα μία αναφορά σε αντικείμενο της κλάσης, π.χ.
    ```
    MyClass& operator+(const MyClass& rhs);
    ```
  - Μέσω μίας μεθόδου που δεν ανήκει στην κλάση και δέχεται δύο ορίσματα. Τουλάχιστον το δεύτερο όρισμα είναι μία αναφορά σε αντικείμενο της κλάσης. Για να μπορέσουν οι μέθοδοι αυτές να έχουν πρόσβαση στα ιδιωτικά δεδομένα της κλάσης, ορίζονται ως φίλες μέσω της λέξης friend, π.χ.
    ```
    friend std::ostream& operator<<(std::ostream& s, const MyClass& rhs);
    ```
- Εφόσον ένα αναφερόμενο αντικείμενο δεν αλλάζει κατά την εφαρμογή τού τελεστή, θα πρέπει να δηλωθεί ρητά πως παραμένει σταθερό, μέσω της δεσμευμένης λέξης **const**.

Παραδείγματα υπερφόρτωσης τελεστών

- Θα χρησιμοποιήσουμε την κλάση Ratio που έχουμε ήδη μελετήσει, η οποία περιγράφει ρητούς αριθμούς, δηλαδή κλάσματα με αριθμητή και παρανομαστή ακεραίους αριθμούς.
- Σε προηγούμενη άσκηση υλοποιήθηκε μία μέθοδος (η Ratio::add), για την πρόσθεση δύο κλασμάτων.
- Θα ήταν όμως καλύτερα να ορίζαμε την πράξη `'+'` για τα αντικείμενα της κλάσης Ratio, έτσι ώστε να μπορούμε π.χ. να γράψουμε τον κώδικα:

```
1 Ratio  a(3,4),  b(4,7),  c;
2 c=a+b;  // εδώ το "+" θα πρέπει να έχει υπερφορτωθεί καταλλήλως
3 cout << "α+β=";   c.print();   cout << endl;
```

- Επίσης, η γραμμή 3 του παραπάνω κώδικα θα ήταν πολύ πιο εύκολα αναγνώσιμη ως

```
cout << "α+β=" << c << endl;
```

αρκεί να υπερφορτώναμε καταλλήλως τον τελεστή `'<<'`, ώστε να μπορεί να χρησιμοποιεί αντικείμενα της κλάσης Ratio.

Παραδείγματα υπερφόρτωσης τελεστών


```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();               // κατασκευή
7      Ratio(int);            // κατασκευή
8      Ratio(int,int);        // κατασκευή
9      ~Ratio();              // καταστροφή
10     int getNumerator()   const;  // πρόσβαση
11     int getDenominator() const;  // πρόσβαση
12     void setNumerator(int);      // πρόσβαση
13     void setDenominator(int);    // πρόσβαση
14     double toDouble();     // πραγματική τιμή
15     void invert();         // αντιστροφή
16     void print();          // εκτύπωση
17     void add(const Ratio&, Ratio&) const;
18
19     // υπερφόρτωση τελεστή +
20     Ratio& operator+(const Ratio& rhs);
21
22     // υπερφόρτωση τελεστή <<
23     friend std::ostream& operator<<(std::ostream& stream,
24                                     const Ratio& rhs);
25
26   private:
27     int num;      // αριθμητής
28     int den;      // παρανομαστής
29 };
```

- Αρχείο `Ratio.h` για τη δήλωση της κλάσης Ratio.
- Στη γραμμή 20, έχει προστεθεί η υπογραφή του υπερφορτωμένου τελεστή `+`.
- Στις γραμμές 23–24, έχει προστεθεί η υπογραφή του υπερφορτωμένου τελεστή `<<`.
  Η λέξη friend δηλώνει πως η υπογραφή αυτή δεν ανήκει στην παρούσα κλάση (Ratio), όμως τής επιτρέπεται (ως φίλης) η πρόσβαση στα ιδιωτικά μέλη της Ratio.

Παραδείγματα υπερφόρτωσης τελεστών


```
1  #include <iostream>     // αρχείο Ratio.cpp
2  #include "Ratio.h"
3  Ratio::Ratio()              { num=0; den=1; }
4  Ratio::Ratio(int i)         { num=i; den=1; }
5  Ratio::Ratio(int i,int j) { num=i; den=j; }
6  Ratio::~Ratio() { }
7  int Ratio::getNumerator()   const  { return num; }
8  int Ratio::getDenominator() const  { return den; }
9  void Ratio::setNumerator(int i)    { num=i; }
10 void Ratio::setDenominator(int i)  { den=i; }
11 double Ratio::toDouble() { return (double) num / (double) den; }
12 void Ratio::invert() { int tmp=num;  num=den;  den=tmp; }
13 void Ratio::print()  { std::cout << num << "/" << den; }
14
15 void Ratio::add(const Ratio& b, Ratio& c) const {
16   int num2=b.getNumerator();
17   int den2=b.getDenominator();
18   int num3=num*den2+num2*den;
19   int den3=den*den2;
20   c.setNumerator(num3);
21   c.setDenominator(den3);
22 }
23
24 Ratio& Ratio::operator+(const Ratio& rhs) {
25   //  int n2=rhs.getNumerator();
26   //  int d2=rhs.getDenominator();
27   //  int n3=num*d2+n2*den;
28   //  int d3=den*d2;
29   //  num=n3;
30   //  den=d3;
31   (*this).add(rhs,*this); // χρήση της μεθόδου add που υλοποιήθηκε
32   return *this;
33 }
34
35 std::ostream& operator<<(std::ostream& stream, const Ratio& rhs) {
36   stream << rhs.num << "/" << rhs.den;
37   return stream;
38 }
```

- Αρχείο `Ratio.cpp` για την υλοποίηση της κλάσης Ratio.

- Στις γραμμές 35–38, υλοποιείται ο υπερφορτωμένος τελεστής **<<**.
- Ο τελεστής **<<** δεν ανήκει στην κλάση Ratio, και για το λόγο αυτό αναφέρεται ως operator<< και όχι ως Ratio::operator<< (αντίθετα με την περίπτωση του τελεστή + στη γραμμή 24).
- Γραμμή 36: Υλοποίηση του τελεστή **<<**, με εγγραφή δεδομένων στο stream (που βρίσκεται αριστερά του **<<**), έχοντας πρόσβαση στα ιδιωτικά μέλη του αντικειμένου rhs (που βρίσκεται δεξιά του **<<**).
- Γραμμή 37: επιστρέφεται αντικείμενο stream που έχει στο μεταξύ μεταβληθεί.

Παραδείγματα υπερφόρτωσης τελεστών


```
1  #include <iostream>     // αρχείο Ratio.cpp
2  #include "Ratio.h"
3  Ratio::Ratio()              { num=0; den=1; }
4  Ratio::Ratio(int i)         { num=i; den=1; }
5  Ratio::Ratio(int i,int j) { num=i; den=j; }
6  Ratio::~Ratio() { }
7  int Ratio::getNumerator()   const  { return num; }
8  int Ratio::getDenominator() const  { return den; }
9  void Ratio::setNumerator(int i)    { num=i; }
10 void Ratio::setDenominator(int i)  { den=i; }
11 double Ratio::toDouble() { return (double) num / (double) den; }
12 void Ratio::invert() { int tmp=num;  num=den;  den=tmp; }
13 void Ratio::print()  { std::cout << num << "/" << den; }
14
15 void Ratio::add(const Ratio& b, Ratio& c) const {
16   int num2=b.getNumerator();
17   int den2=b.getDenominator();
18   int num3=num*den2+num2*den;
19   int den3=den*den2;
20   c.setNumerator(num3);
21   c.setDenominator(den3);
22 }
23
24 Ratio& Ratio::operator+(const Ratio& rhs) {
25   //  int n2=rhs.getNumerator();
26   //  int d2=rhs.getDenominator();
27   //  int n3=num*d2+n2*den;
28   //  int d3=den*d2;
29   //  num=n3;
30   //  den=d3;
31   (*this).add(rhs,*this); // χρήση της μεθόδου add που υλοποιήθηκε
32   return *this;
33 }
34
35 std::ostream& operator<<(std::ostream& stream, const Ratio& rhs) {
36   stream << rhs.num << "/" << rhs.den;
37   return stream;
38 }
```

- Στις γραμμές 24–33, υλοποιείται ο υπερφορτωμένος τελεστής +.
  - Ο δείκτης `this` δείχνει στο παρόν αντικείμενο (αριστερά του +).
  - `*this` : το παρόν αντικείμενο
  - Γραμμή 31: Καλείται μέσω του παρόντος αντικειμένου η μέθοδος add, που προσθέτει το rhs (δεξιά του +) στο παρόν αντικείμενο (αριστερά του +). Το δε αποτέλεσμα αποθηκεύεται στο παρόν αντικείμενο.
  - Γραμμή 32: επιστρέφεται το παρόν αντικείμενο που έχει στο μεταξύ μεταβληθεί.
- Εναλλακτικά, αντί της γραμμής 31 θα μπορούσαν να χρησιμοποιηθούν οι γραμμές 25–30, που ουσιαστικά είναι όμοιες με τις γραμμές 16–21 της μεθόδου add.

Παραδείγματα υπερφόρτωσης τελεστών


```cpp
#include <iostream>     // αρχείο Ratio.cpp
#include "Ratio.h"
Ratio::Ratio()              { num=0; den=1; }
Ratio::Ratio(int i)         { num=i; den=1; }
Ratio::Ratio(int i,int j) { num=i; den=j; }
Ratio::~Ratio() { }
int Ratio::getNumerator()   const  { return num; }
int Ratio::getDenominator() const  { return den; }
void Ratio::setNumerator(int i)    { num=i; }
void Ratio::setDenominator(int i)  { den=i; }
double Ratio::toDouble() { return (double) num / (double) den; }
void Ratio::invert() { int tmp=num;  num=den;  den=tmp; }
void Ratio::print()  { std::cout << num << "/" << den; }

void Ratio::add(const Ratio& b, Ratio& c) const {
  int num2=b.getNumerator();
  int den2=b.getDenominator();
  int num3=num*den2+num2*den;
  int den3=den*den2;
  c.setNumerator(num3);
  c.setDenominator(den3);
}

Ratio& Ratio::operator+(const Ratio& rhs) {
  //  int n2=rhs.getNumerator();
  //  int d2=rhs.getDenominator();
  //  int n3=num*d2+n2*den;
  //  int d3=den*d2;
  //  num=n3;
  //  den=d3;
  (*this).add(rhs,*this); // χρήση της μεθόδου add που υλοποιήθηκε
  return *this;
}

std::ostream& operator<<(std::ostream& stream, const Ratio& rhs) {
  stream << rhs.num << "/" << rhs.den;
  return stream;
}
```

- Η υλοποίηση αυτή του τελεστή + παρουσιάζει προβλήματα.

  Αν x, y και z είναι αντικείμενα της κλάσης Ratio, τότε
  - Μπορεί να υπολογιστεί το **z=x+y;**
    όμως τα δεδομένα του x αλλάζουν μετά την πράξη!!!
  - Μπορεί να υπολογιστεί το **z=y+1;**
    όμως τα δεδομένα του y αλλάζουν μετά την πράξη!!!
  - Λάθος μεταγλώττισης για το **z=1+y;**
    όπου το + αναμένεται ως τελεστής άθροισης ακεραίων, λόγω του ακεραίου αριστερού μέλους.
- Για την αποφυγή/διόρθωση των προβλημάτων αυτών μπορούμε να χρησιμοποιήσουμε για τον τελεστή + μία φίλη μέθοδο αντί της μεθόδου μέλους.

Παραδείγματα υπερφόρτωσης τελεστών

Αλλαγές για το αρχείο Ratio.h :

```
1 // αλλαγή της γραμμής 20 του αρχείου Ratio.h σε:
2 friend Ratio operator+(const Ratio& lhs, const Ratio& rhs);
```

Αλλαγές για το αρχείο Ratio.cpp :

```
1 // αλλαγή των γραμμών 24--33 του αρχείου Ratio.cpp σε:
2 Ratio operator+(const Ratio& lhs, const Ratio& rhs) {
3   Ratio res;          // δημιουργία του αντικειμένου res
4   lhs.add(rhs,res);   // lhs + rhs --> αποθήκευση στο res
5   return res;         // επιστροφή του res
6 }
```

Αρχείο testRatio.cpp για τον έλεγχο της κλάσης:

```
1  #include <iostream>
2  #include "Ratio.h"
3  using namespace std;
4
5  int main() {
6    Ratio x(1,2), y(1,2), z(1,2);
7    cout << "x=" << x << " y=" << y << " z=" << z << endl;
8    cout << "x+y+z=" << (x+y+z) << endl;
9    cout << "x=" << x << " y=" << y << " z=" << z << endl;
10   cout << "y+1=" << (y+1) << endl;
11   cout << "x=" << x << " y=" << y << " z=" << z << endl;
12
13   //--- χρησιμοποιούνται με τη σωστή υλοποίηση του +
14   cout << "1+z=" << (1+z) << endl;    //   <--- g++ errors...
15   cout << "x=" << x << " y=" << y << " z=" << z << endl;
16 }
```

- Για την άρση των προβλημάτων που προκύπτουν με την προηγούμενη υλοποίηση του τελεστή +, κάνουμε τις διπλανές αλλαγές στα αρχεία Ratio.h και Ratio.cpp.
- Έξοδος προβληματικής υλοποίησης του + :

```
x=1/2 y=1/2 z=1/2
x+y+z=12/8
x=12/8 y=1/2 z=1/2
y+1=3/2
x=12/8 y=3/2 z=1/2
// 1+z : g++ errors
```

- Έξοδος σωστής υλοποίησης του + :

```
x=1/2 y=1/2 z=1/2
x+y+z=12/8
x=1/2 y=1/2 z=1/2
y+1=3/2
x=1/2 y=1/2 z=1/2
1+z=3/2
x=1/2 y=1/2 z=1/2
```

Παράδειγμα υπερφόρτωσης τελεστή +=

Προσθήκη στο αρχείο Ratio.h :

```
1  Ratio& operator+=(const Ratio& rhs);
```

Προσθήκη στο αρχείο Ratio.cpp :

```
1  Ratio& Ratio::operator+=(const Ratio& rhs) {
2    (*this).add(rhs,*this);
3    return *this;
4  }
```

Προσθήκη στο αρχείο testRatio.cpp για τον έλεχγο της κλάσης:

```
1    Ratio q1(1,2),q2(1,2), q3(1,2);
2    cout << "q1=" << q1 << " q2=" << q2 << " q3=" << q3 << endl;
3    q1+=q2+q3;
4    cout << "q1+=q2+q3 => q1=" << q1 << endl;
5    cout << "q1=" << q1 << " q2=" << q2 << " q3=" << q3 << endl;
6    q1+=10+q2;
7    cout << "q1+=10+q2 => q1=" << q1 << endl;
8    cout << "q1=" << q1 << " q2=" << q2 << " q3=" << q3 << endl;
9    q1+=q3+4;
10   cout << "q1+=q3+4 => q1=" << q1 << endl;
11   cout << "q1=" << q1 << " q2=" << q2 << " q3=" << q3 << endl;
```

- Η «προβληματική» υλοποίηση του τελεστή +, που αλλάζει τα δεδομένα του αριστερού τελεστέου, είναι μάλλον ιδανική για την υλοποίηση του τελεστη +=.
- Στην περίπτωση αυτή θέλουμε να προστεθεί το δεξί μέλος στην υπάρχουσα τιμή του αριστερού μέλους.
- Έξοδος προγράμματος ελέγχου :

```
q1=1/2 q2=1/2 q3=1/2
q1+=q2+q3 => q1=12/8
q1=12/8 q2=1/2 q3=1/2
q1+=10+q2 => q1=192/16
q1=192/16 q2=1/2 q3=1/2
q1+=q3+4 => q1=528/32
q1=528/32 q2=1/2 q3=1/2
```