Κατασκευαστής αντιγραφής και αντιγραφή ανάθεσης

#12

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

Συναρτήσεις που δημιουργούνται αυτόματα

- Οι ακόλουθες δύο συναρτήσεις μέλη που έχουμε ήδη συναντήσει, κάτω από ορισμένες περιστάσεις δημιουργούνται αυτόματα από τον μεταγλωττιστή
  - **Κατασκευαστής**: Ένας κενός προκαθορισμένος κατασκευαστής (δηλαδή ένας κατασκευαστής χωρίς παραμέτρους και με κενό σώμα) δημιουργείται αυτόματα αν δεν οριστεί κάποιος άλλος κατασκευαστής.
  - **Καταστροφέας**: Ένας κενός καταστροφέας δημιουργείται αν δεν οριστεί καταστροφέας (επίσης με κενό σώμα).
- Δύο άλλες συναρτήσεις μέλη που επίσης, κάτω από ορισμένες περιστάσεις δημιουργούνται αυτόματα είναι ο κατασκευαστής αντιγραφής (**copy constructor**) και ο τελεστής ανάθεσης (**assignment operator**)

Κατασκευαστής αντιγραφής (copy constructor)

- Ο κατασκευαστής αντιγραφής είναι κατασκευαστής, και συνεπώς:
  - Έχει το ίδιο όνομα με την κλάση.
  - Δεν έχει τύπο επιστροφής (αν και φαίνεται σαν να επιστρέφει ένα αντικείμενο της κλάσης του όταν καλείται ρητά).
- Όπως και με τον κατασκευαστή μετατροπής υπάρχουν περιπτώσεις που ο copy constructor καλείται υπονοούμενα. Αυτό συμβαίνει:
  - Όταν ένα αντικείμενο δηλώνεται να έχει την ίδια τιμή με ένα άλλο αντικείμενο
    - Παράδειγμα:

```
Fraction f1(1,2);
Fraction f2 = f1; // το νέο αντικείμενο αρχικοποιείται ως ένα αντίγραφο του f1
```

  - Όταν ένα αντικείμενο **περνά με τιμή (by value)** σε μια συνάρτηση
  - Όταν ένα αντικείμενο **επιστρέφεται με τιμή (by value)** από μια συνάρτηση

Δήλωση κατασκευαστή αντιγραφής

- Καθώς ο σκοπός ενός κατασκευαστή αντιγραφής είναι να αρχικοποιηθεί ένα νέο αντικείμενο ως αντίγραφο ενός άλλου αντικειμένου, δέχεται ένα αντικείμενο ως παράμετρο.
  - `classname(const classname&)`
- Η παράμετρος είναι `const` (αν και δεν απαιτείται) διότι ο κατασκευαστής αντιγραφής **δεν θα πρέπει** να τροποποιεί την παράμετρο.
- Η παράμετρος θα πρέπει να περνά με αναφορά. Αν περνούσε με τιμή τότε λόγω αναδρομικής κλήσης της ίδιας συνάρτησης, δηλαδή του copy constructor θα οδηγούμασταν σε stack overflow (ωστόσο, ο μεταγλωττιστής δεν επιτρέπει να δηλώσουμε σε κατασκευαστή αντιγραφής την παράμετρο να περνά με τιμή).
- Παραδείγματα:
  - `Fraction(const Fraction &f);`
  - `Mixed(const Mixed& m);`

Κατασκευαστής αντιγραφής

- Τι θα πρέπει να κάνει ένας copy constructor;
  - Να αντιγράφει τα μέλη δεδομένων από την παράμετρο στο νέο αντικείμενο.
  - Παράδειγμα:

```
Fraction(const Fraction &f) {
  numer = f.numer;
  denom = f.denom;
}
```

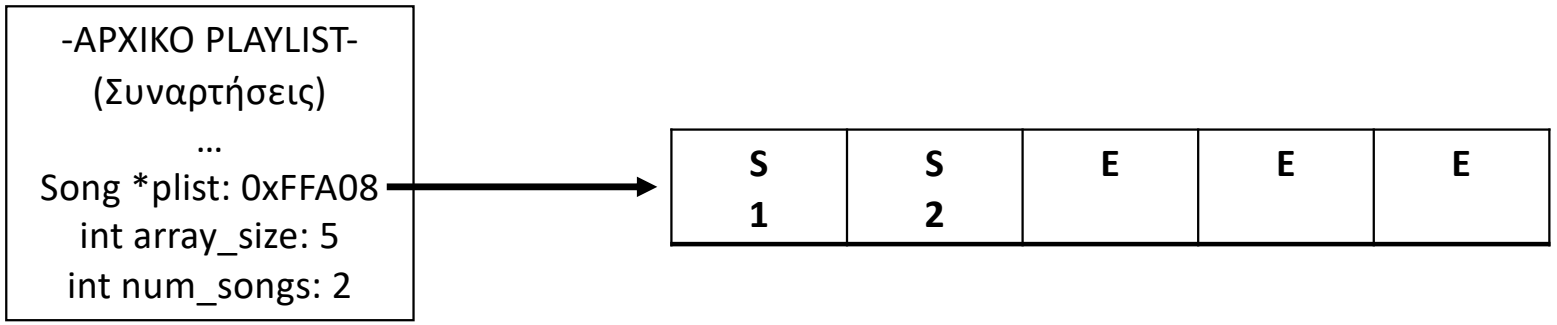
ή

```
Fraction(const Fraction &f) {
  this->numer = f.numer;
  this->denom = f.denom;
}
```

  - Ο copy constructor που δημιουργείται αυτόματα (default copy constructor) κάνει ακριβώς αυτό.
  - Ωστόσο, δεν είναι αυτή η συμπεριφορά που θα θέλαμε σε όλες τις περιπτώσεις

«Προβληματικός» προκαθορισμένος κατασκευαστής αντιγραφής: ρηχή αντιγραφή

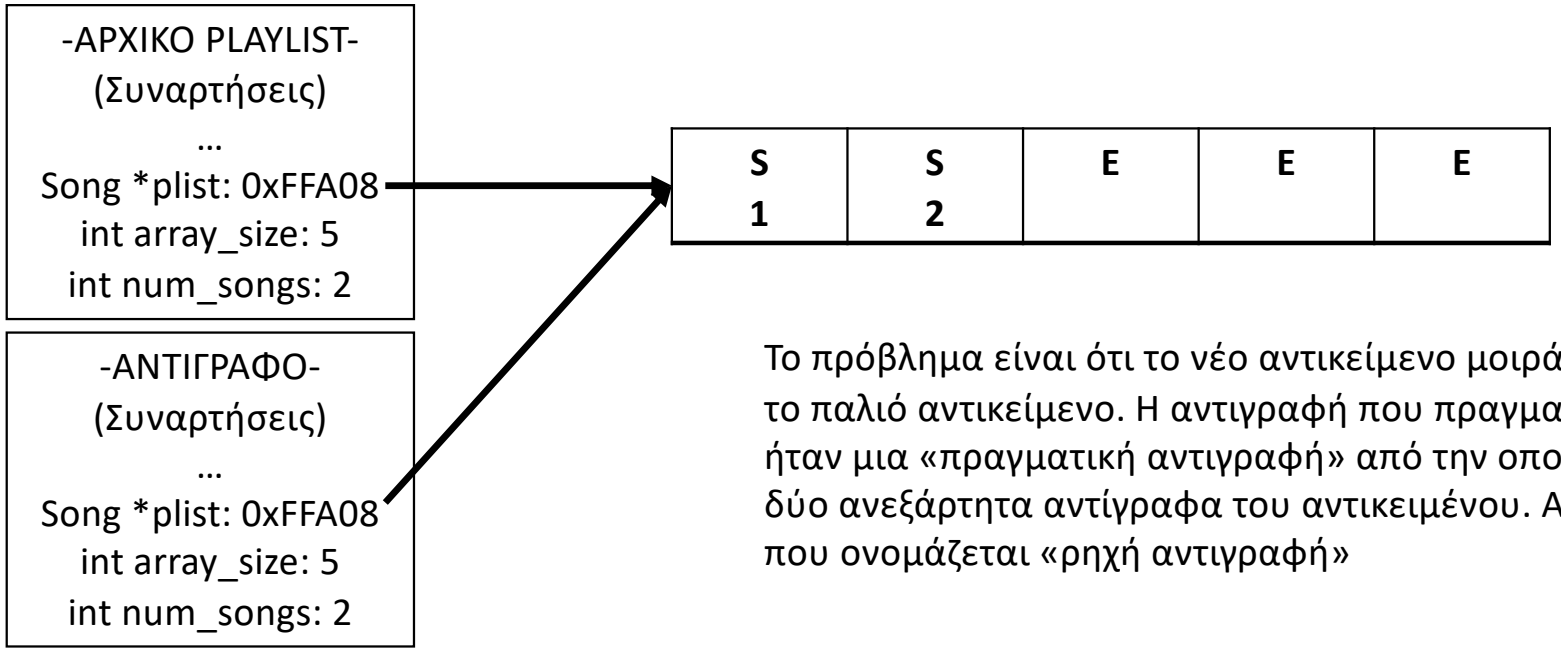
- Έστω ότι θέλουμε να αντιγράψουμε ένα αντικείμενο playlist



- Η ρηχή αντιγραφή (shallow copy) κάνει **ακριβής** αντιγραφή όλων των μελών δεδομένων από το παλιό αντικείμενο στο νέο

Λειτουργία της προκαθορισμένης ρηχής αντιγραφής

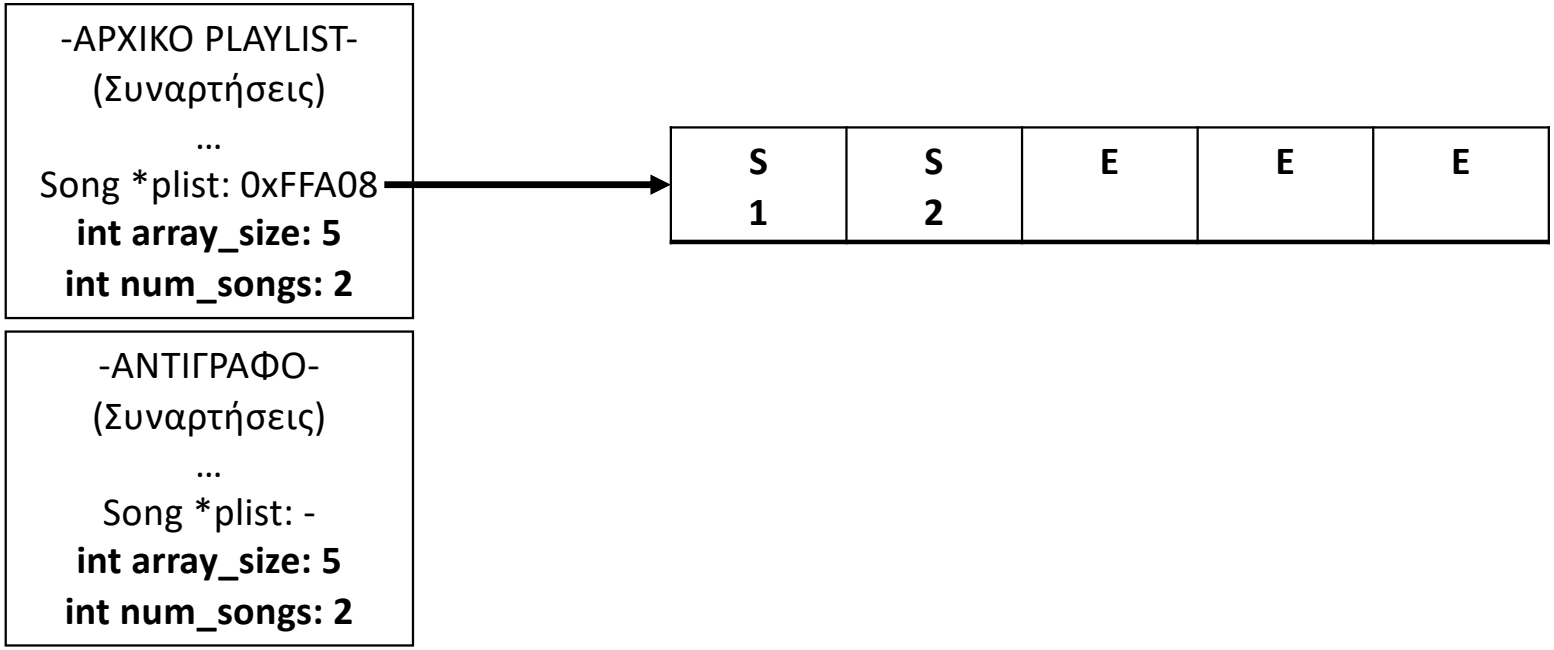
- Στον κατασκευαστή αντιγραφής το αρχικό αντικείμενο είναι η παράμετρος.
- Στο αντίγραφο ορίζονται τα μέλη δεδομένων να είναι ίδια με αυτά του αρχικού αντικειμένου.



Το πρόβλημα είναι ότι το νέο αντικείμενο μοιράζεται την plist με το παλιό αντικείμενο. Η αντιγραφή που πραγματοποιήθηκε δεν ήταν μια «πραγματική αντιγραφή» από την οποία θα προκύπτανε δύο ανεξάρτητα αντίγραφα του αντικειμένου. Αυτός είναι ο λόγος που ονομάζεται «ρηχή αντιγραφή»

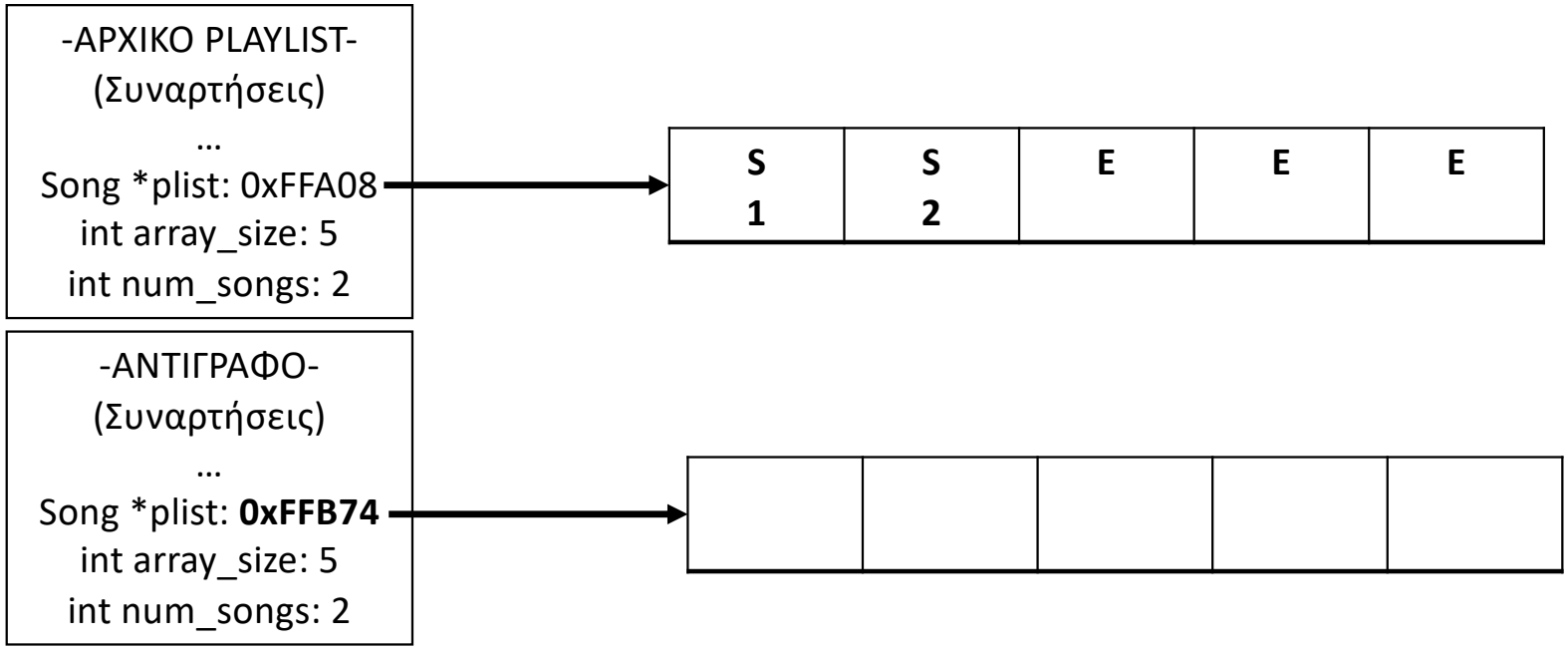
Βαθιά αντιγραφή (πρέπει να υλοποιηθεί ως κατασκευαστής αντιγραφής από το χρήστη)

- Ο κατασκευαστής αντιγραφής έχει το αρχικό αντικείμενο ως παράμετρο.
- Τα μέλη δεδομένα που δεν είναι δείκτες ορίζονται να έχουν ίσες τιμές με τα μέλη δεδομένων του αρχικού αντικειμένου



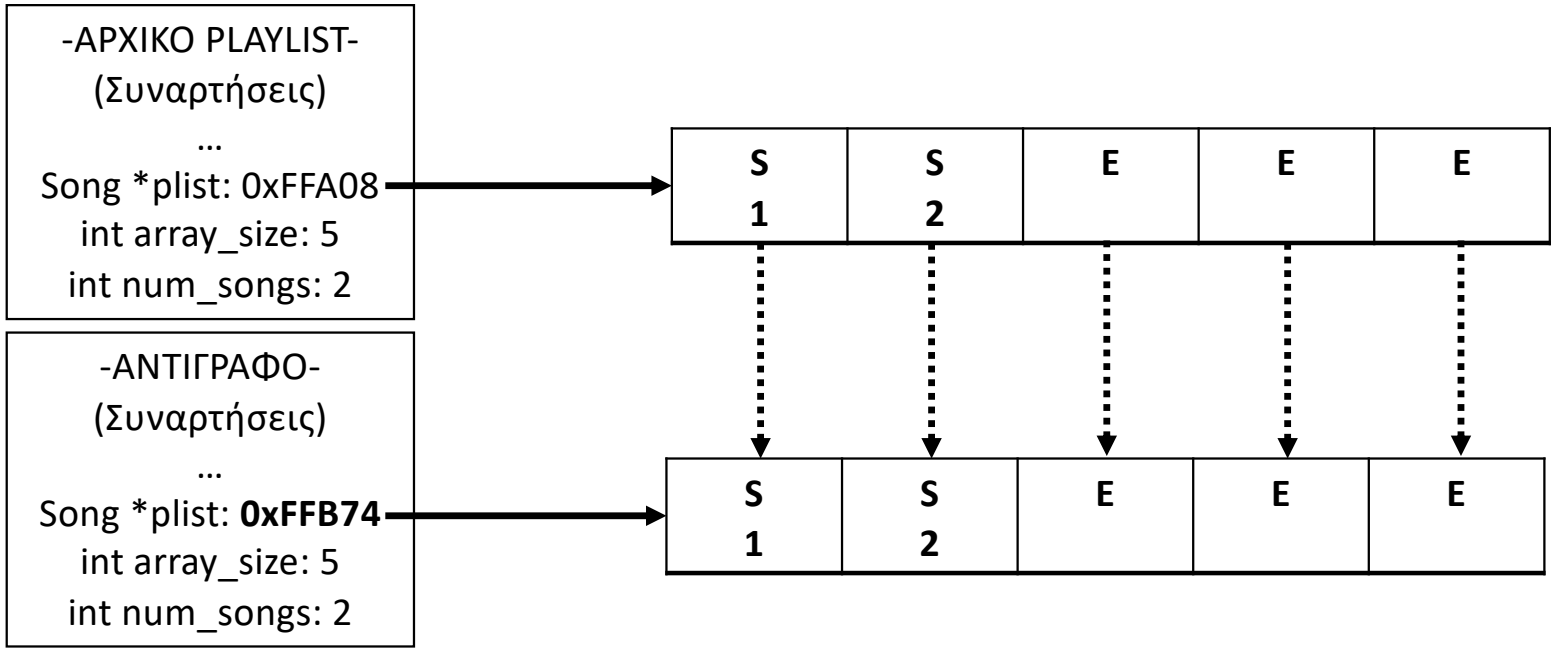
Βαθιά αντιγραφή (deep copy)

- Δεσμεύεται νέα μνήμη στην οποία θα δείχνει ο δείκτης του αντιγράφου.



Βαθιά αντιγραφή (deep copy)

- Αντιγράφονται τα δεδομένα από την παλιά δυναμική μνήμη στη νέα.



Τελεστής ανάθεσης (assignment operator)

- Ο τελεστής ανάθεσης (=) καλείται όταν ένα αντικείμενο εκχωρείται σε ένα άλλο.
- Μοιάζει με τον κατασκευαστή αντιγραφής αλλά υπάρχουν ορισμένες βασικές διαφορές:
  - Ο τελεστής ανάθεσης είναι μια κανονική συνάρτηση μέλος της κλάσης και όχι ένας κατασκευαστής, άρα αφορά δύο αντικείμενα που ήδη υπάρχουν και έχουν αρχικοποιηθεί.
  - Ο τελεστής ανάθεσης επιστρέφει την τιμή που του ανατίθεται (έτσι, επιτρέπονται και διαδοχικές (cascading) κλήσεις του τελεστή ανάθεσης)

```
Fraction f1(1,2), f2, f3;
f3 = f2 = f1; // τα αντικείμενα f2 και f3 λαμβάνουν την ίδια τιμή.
```

Τελεστής ανάθεσης

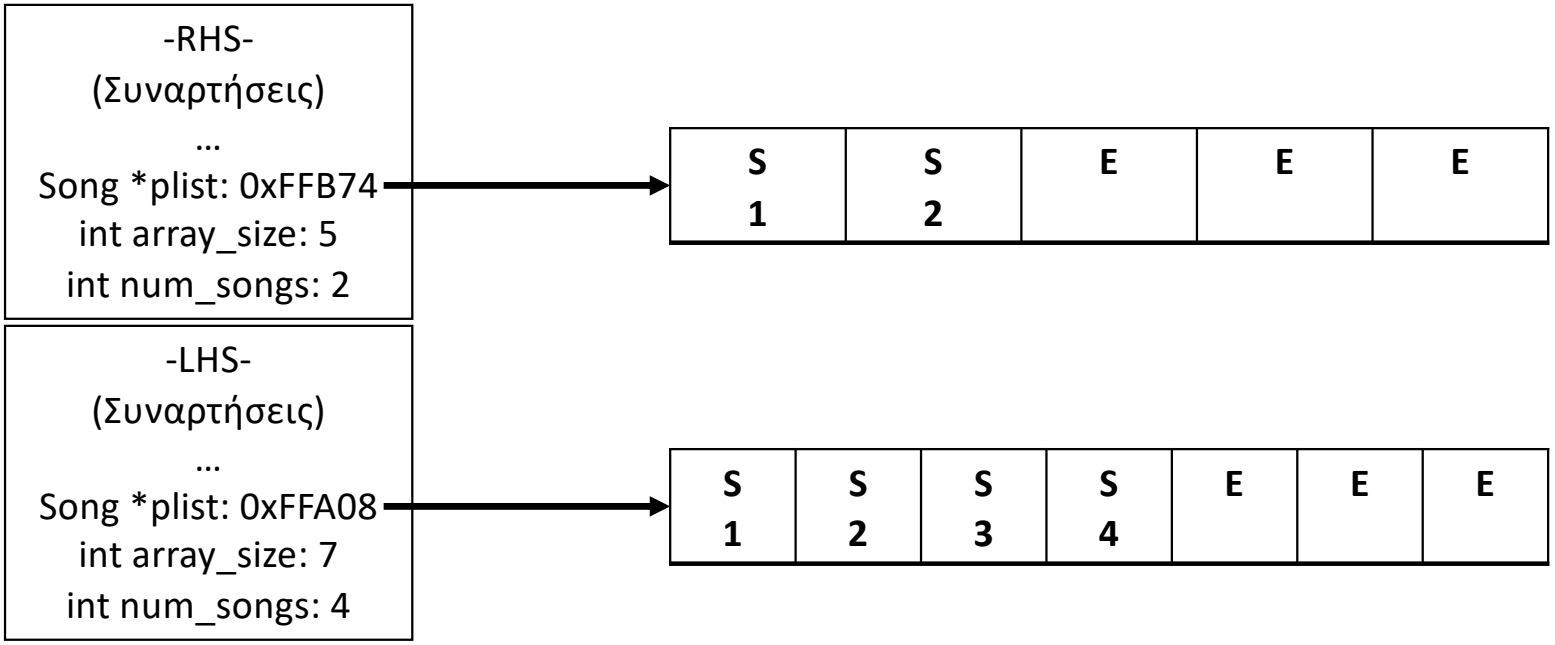
- Μορφή:
  - `classname& operator=(const classname&)`
- Παράδειγμα:
  - ```
    Fraction lhs(1,2), rhs(2,5);
    lhs = rhs;
    ```
  - `lhs` είναι το καλών αντικείμενο, ενώ `rhs` είναι η παράμετρος, η συνάρτηση του τελεστή ανάθεσης αλλάζει το `lhs` έτσι ώστε να είναι ένα αντίγραφο του `rhs` και επιστρέφει μια αναφορά στο `lhs` μέσω του δείκτη `this`.
    ```

Ο δείκτης `this`

- Σε κάθε αντικείμενο υπάρχει ένας δείκτης που ονομάζεται `this`
- Είναι σαν να υπάρχει η ακόλουθη δήλωση στα μέλη δεδομένων του αντικειμένου:
  - `classname *this;`
- Ο δείκτης `this` δείχνει προς το ίδιο το αντικείμενο.
- Μπορούμε να καλούμε άλλες συναρτήσεις μέλη με την εντολή:
  - `this->memberFunction();`
- Χρησιμοποιώντας το δείκτη `this` επιστρέφουμε μια αναφορά προς το ίδιο το αντικείμενο στον τελεστή ανάθεσης.

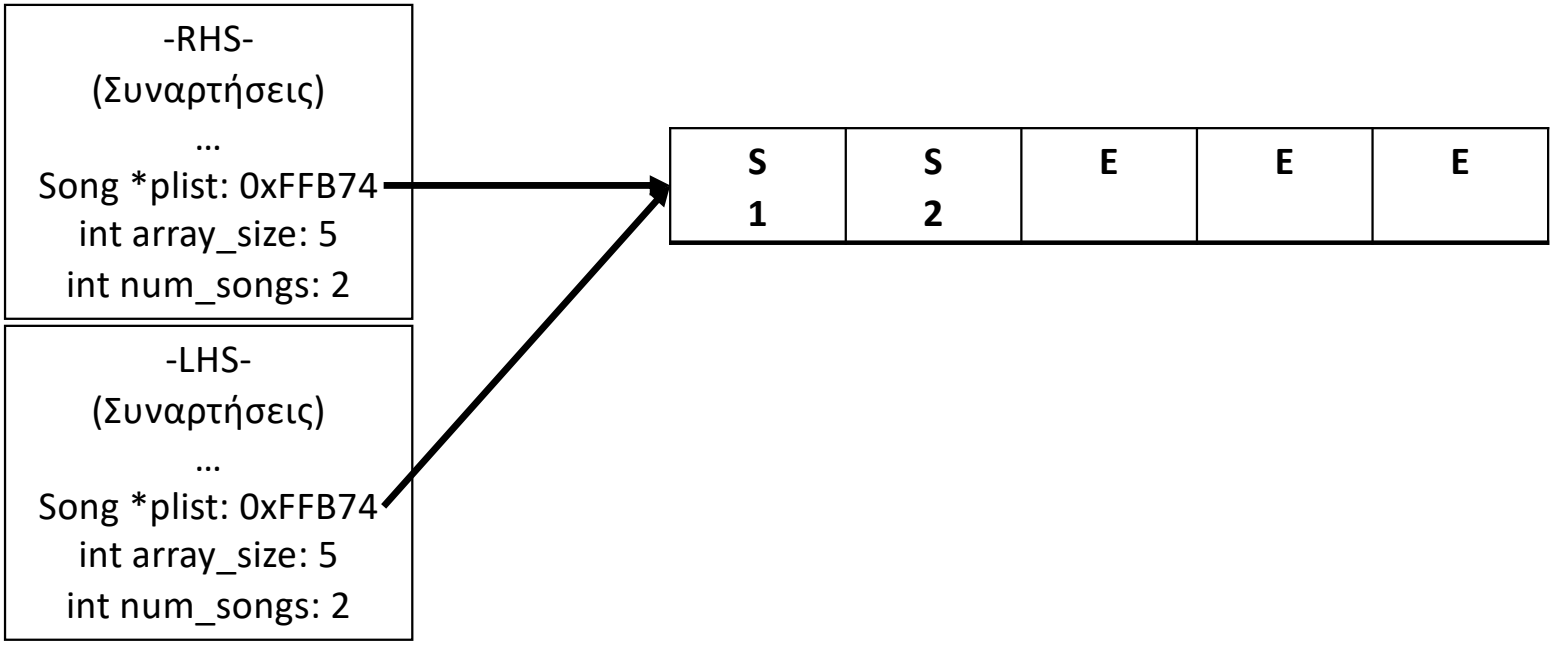
Ρηχή αντιγραφή με ανάθεση

- Έστω ότι αναθέτουμε το αντικείμενο playlist RHS στο LHS.
  - `LHS = RHS;`



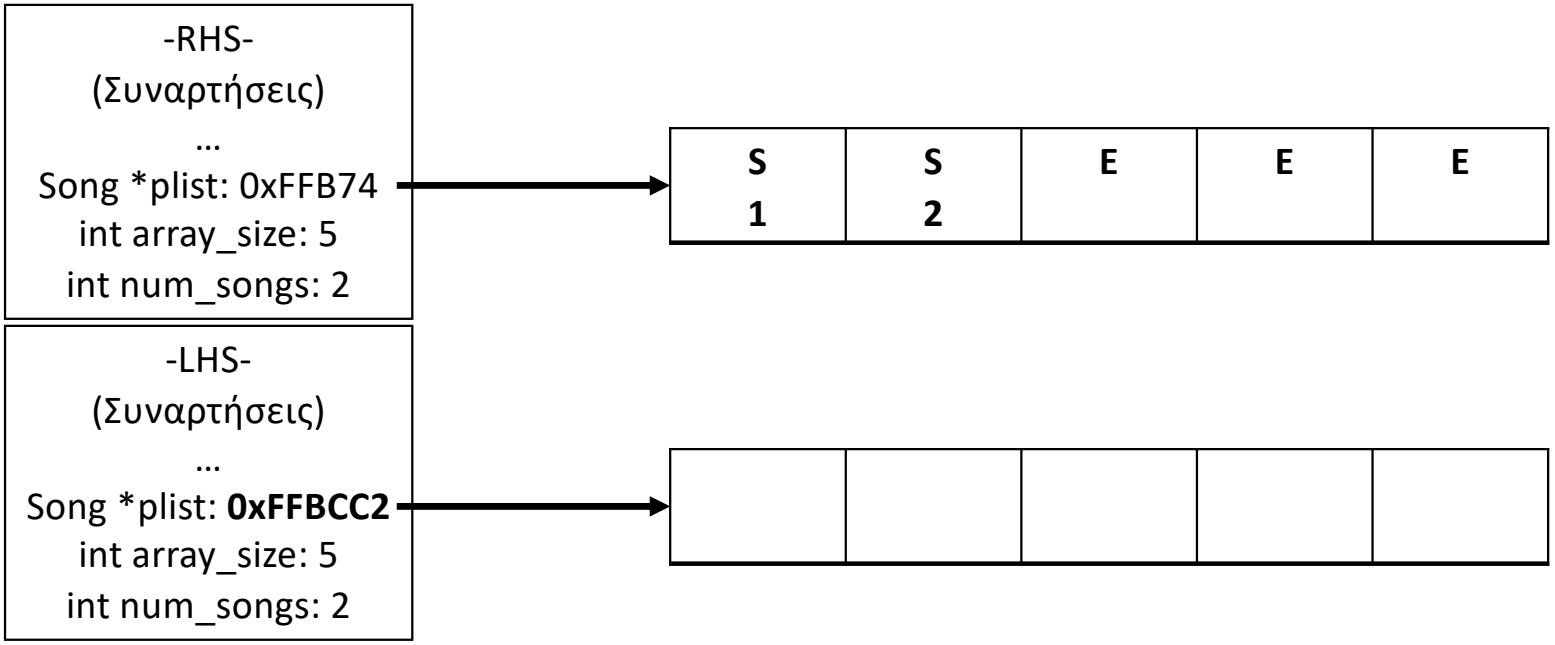
Ρηχή αντιγραφή με ανάθεση

- `LHS = RHS;`
- Ο τελεστής ανάθεσης **που αυτόματα δημιουργείται** από το μεταγλωττιστή πραγματοποιεί ρηχή αντιγραφή.



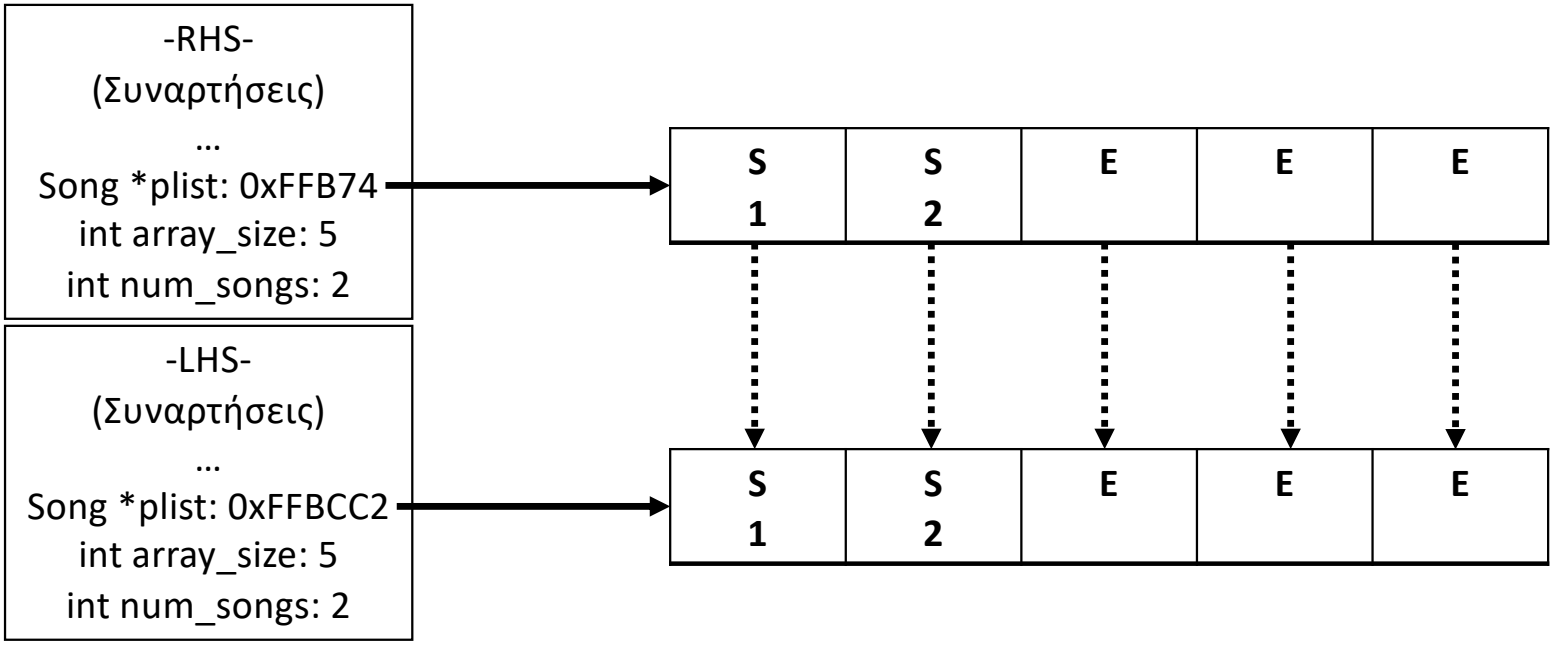
Βαθιά αντιγραφή με ανάθεση

- `LHS = RHS;`
- Καθώς ο πίνακας που έχει δεσμεύσει το LHS για `plist` δεν έχει το σωστό μέγεθος, θα πρέπει να αποδεσμευθεί η μνήμη του και να δεσμευθεί μνήμη κατάλληλου μεγέθους.



Βαθιά αντιγραφή με ανάθεση

- `LHS = RHS;`
- Στη συνέχεια θα πρέπει να αντιγραφούν τα στοιχεία του `plist` του RHS στο LHS, και επιπλέον να αντιγραφούν τα υπόλοιπα μέλη δεδομένων.



Κατασκευαστής αντιγραφής και αντιγραφή ανάθεσης

- Αν οριστεί κατασκευαστής αντιγραφής, και δεν οριστεί κάποιος άλλος κατασκευαστής, ο μεταγλωττιστής **δεν θα δημιουργήσει** προκαθορισμένο κατασκευαστή.
- Ο τελεστής ανάθεσης θα πρέπει να είναι συνάρτηση μέλος και όχι friend συνάρτηση.
- Ο τελεστής ανάθεσης ονομάζεται αλλιώς και τελεστής αντιγραφής και υλοποιεί την αντιγραφή ανάθεσης (copy assignment).
- Ο τελεστής ανάθεσης πάντα έχει ως τελευταία εντολή:

```
return *this;
```

Κώδικας song και playlist για shallow copy και deep copy

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/playlist_shallow_copy.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/playlist_deep_copy.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/playlist_deleted.cpp

Άσκηση #1: Διάφοροι κατασκευαστές και τελεστής ανάθεσης

- Ορίστε μια κλάση Distance που να αναπαριστά απόσταση σε μέτρα έχοντας ένα ιδιωτικό μέλος με όνομα meters. Η κλάση να διαθέτει:
  - **Κατασκευαστή χωρίς παραμέτρους (default constructor)**, ο οποίος αρχικοποιεί την απόσταση σε 0 μέτρα. Επιπλέον να εμφανίζει μήνυμα ότι κλήθηκε ο συγκεκριμένος κατασκευαστής, όταν αυτό συμβαίνει.
  - **Κατασκευαστές μετατροπής**, ώστε να μπορεί να δημιουργηθεί αντικείμενο Distance είτε από έναν ακέραιο αριθμό που υποδηλώνει μέτρα (π.χ. Distance d = 400; θα σημαίνει 400 μέτρα) είτε από έναν αριθμό κινητής υποδιαστολής που εκφράζει χιλιόμετρα (π.χ. Distance d = 3.5; θα σημαίνει 3500 μέτρα). Επιπλέον να εμφανίζει μήνυμα ότι κλήθηκε ο συγκεκριμένος κατά περίπτωση κατασκευαστής, όταν αυτό συμβαίνει.
  - **Κατασκευαστή αντιγραφής (copy constructor)** για τη δημιουργία νέου αντικειμένου ως αντίγραφο άλλου αντικειμένου Distance. Επιπλέον να εμφανίζει μήνυμα ότι κλήθηκε ο συγκεκριμένος κατασκευαστής , όταν αυτό συμβαίνει.
  - **Τελεστή ανάθεσης αντιγραφής (operator=)** για την αντιγραφή της τιμής ενός αντικειμένου σε ένα ήδη υπάρχον άλλο αντικείμενο. Επιπλέον να εμφανίζει μήνυμα ότι έγινε κλήση του συγκεκριμένου τελεστή, όταν αυτό συμβαίνει.
  - **Υπερφόρτωση του τελεστή <<** για εμφάνιση αντικειμένων Distance με το std::cout.
- Για τον ακόλουθο κώδικα σημειώστε ποιος κατασκευαστής ή ποιος τελεστής ανάθεσης καλείται σε κάθε περίπτωση και τι πρόκειται να εκτυπωθεί.

```
int main() {
  Distance d1;
  Distance d2(250);
  Distance d3 = 300;
  Distance d4(2.5);
  Distance d5 = 7.8;
  Distance d6(d1);
  Distance d7 = d2;
  d1 = d3;
  for (const Distance *d : {&d1, &d2, &d3, &d4, &d5, &d6, &d7}) {
    cout << *d << endl;
  }
  return 0;
}
```

Άσκηση #1: Λύση

```cpp
class Distance {
  double meters;

 public:
  Distance() : meters(0) { cout << "Distance() called" << endl; }

  Distance(int m) : meters(m) { cout << "Distance(int) called" << endl; }

  Distance(double km) : meters(km * 1000.0) {
    cout << "Distance(double) called" << endl;
  }

  Distance(const Distance& other) : meters(other.meters) {
    cout << "Distance(const Distance&) called" << endl;
  }

  Distance& operator=(const Distance& other) {
    cout << "operator= called" << endl;
    meters = other.meters;
    return *this;
  }

  friend ostream& operator<<(ostream& os, const Distance& other) {
    os << "Distance: " << other.meters << "m";
    return os;
  }
};
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/exercise1.cpp

```cpp
int main() {
  Distance d1;         // Distance() called
  Distance d2(250);    // Distance(int) called
  Distance d3 = 300;   // Distance(int) called
  Distance d4(2.5);    // Distance(double) called
  Distance d5 = 7.8;   // Distance(double) called
  Distance d6(d1);     // Distance(const Distance&) called
  Distance d7 = d2;    // Distance(const Distance&) called
  d1 = d3;             // operator= called
  for (const Distance *d : {&d1, &d2, &d3, &d4, &d5, &d6, &d7}) {
    cout << *d << endl;
  }
  return 0;
}
```

```
Distance() called
Distance(int) called
Distance(int) called
Distance(double) called
Distance(double) called
Distance(const Distance&) called
Distance(const Distance&) called
operator= called
Distance: 300m
Distance: 250m
Distance: 300m
Distance: 2500m
Distance: 7800m
Distance: 0m
Distance: 250m
```

Άσκηση #2: Ρηχή και βαθιά αντιγραφή

- Να υλοποιηθούν δύο κλάσεις: η κλάση Book, η οποία περιέχει το δημόσιο μέλος string title, και η κλάση Library, η οποία αποθηκεύει έναν vector<Book*> με δείκτες σε βιβλία.
- Η Library να διαθέτει:
  - κατασκευαστή αντιγραφής (με ρηχή αντιγραφή),
  - μέθοδο προσθήκης ενός βιβλίου add_book(Book*)
  - μια μέθοδο print() που εμφανίζει τα στοιχεία των βιβλίων.
- Στη main() να δημιουργηθούν δύο βιβλία, ένα αντικείμενο Library και να προστεθούν σε αυτό τα βιβλία.
- Να δημιουργηθεί ένα επιπλέον αντικείμενο Library, ως αντίγραφο του πρώτου αντικειμένου Library χρησιμοποιώντας τον κατασκευαστή αντιγραφής.
- Αλλάξτε τον τίτλο ενός βιβλίου και εκτυπώστε τα περιεχόμενα των δύο αντικειμένων Library. Πως εκδηλώνεται η ρηχή αντιγραφή που πραγματοποιείται;
- Μετατρέψτε τον κατασκευαστή αντιγραφής έτσι ώστε να πραγματοποιεί βαθιά αντιγραφή.
- Αλλάξτε τον τίτλο ενός βιβλίου και εκτυπώστε τα περιεχόμενα των δύο αντικειμένων Library. Τι παρατηρείτε;

Άσκηση #2: Λύση

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/playlist_deep_copy.cpp

Ρηχή αντιγραφή

```
class Book {
 public:
  string title;
};

class Library {
  vector<Book*> books;

 public:
  Library() {}

  void add_book(Book* b) { books.push_back(b); }

  Library(const Library& other) { books = other.books; }

  void print() const {
    for (auto b : books) cout << b->title << "\n";
  }
};
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect12/exercise2_shallow.cpp

Βαθιά αντιγραφή

```
class Book {
 public:
  string title;
};

class Library {
  vector<Book*> books;

 public:
  Library() {}

  void add_book(Book* b) { books.push_back(b); }

  Library(const Library& other) {
    for (auto b : other.books) {
      Book* newBook = new Book();
      newBook->title = b->title;
      books.push_back(newBook);
    }
  }

  void print() const {
    for (auto b : books) cout << b->title << "\n";
  }
};
```

Ερωτήσεις σύνοψης

- Ποιο είναι το πρωτότυπο του κατασκευαστή αντιγραφής (copy constructor);
- Ποια είναι η διαφορά ανάμεσα στη ρηχή αντιγραφή (shallow copy) και στη βαθιά αντιγραφή (deep copy); Ποια είναι η προκαθορισμένη;
- Πότε καλείται ο κατασκευαστής αντιγραφής για ένα αντικείμενο;
- Γιατί η παράμετρος στον κατασκευαστή αντιγραφής περνά με const αναφορά; Τι θα συνέβαινε αν περνούσε με τιμή;
- Ποιο είναι το πρωτότυπο για τον τελεστή ανάθεσης (assignment operator);
- Ποιες είναι οι διαφορές ανάμεσα στον κατασκευαστή αντιγραφής και στην ανάθεση αντιγραφής;
- Τι θα πρέπει να επιστρέφει ο `operator=`;

Απαντήσεις στις ερωτήσεις σύνοψης

- Ποιο είναι το πρωτότυπο του κατασκευαστή αντιγραφής (copy constructor);
  - ClassName(const ClassName& other);
- Ποια είναι η διαφορά ανάμεσα στη ρηχή αντιγραφή (shallow copy) και στη βαθιά αντιγραφή (deep copy); Ποια είναι η προκαθορισμένη;
  - Στη ρηχή αντιγραφή αντιγράφονται οι τιμές των δεικτών και όχι τα δεδομένα που δείχνουν, ενώ στη βαθιά αντιγραφή δημιουργείται ένα αντίγραφο των δεδομένων που δείχνουν οι δείκτες. Προκαθορισμένη είναι η ρηχή αντιγραφή.
- Πότε καλείται ο κατασκευαστής αντιγραφής για ένα αντικείμενο;
  - Ο κατασκευαστής αντιγραφής καλείται όταν δημιουργείται ένα αντικείμενο ως αντίγραφο ενός άλλου αντικειμένου, όπως για παράδειγμα στο: MyClass a; MyClass b = a;
- Γιατί η παράμετρος στον κατασκευαστή αντιγραφής περνά με const αναφορά; Τι θα συνέβαινε αν περνούσε με τιμή;
  - Με const & απαγορεύεται μέσα στον κατασκευαστή να τροποποιηθεί η παράμετρος, η C++ επιβάλει να περάσει αναφορά καθώς αλλιώς θα οδηγούμασταν σε άπειρο βρόχο (το πέρασμα με τιμή θα καλούσε ξανά τον κατασκευαστή αντιγραφής).
- Ποιο είναι το πρωτότυπο για τον τελεστή ανάθεσης (assignment operator);
  - ClassName(const ClassName& other);
- Ποιες είναι οι διαφορές ανάμεσα στον κατασκευαστή αντιγραφής και στην ανάθεση αντιγραφής (copy assignment);
  - Ο κατασκευαστής αντιγραφής δημιουργεί ένα νέο αντικείμενο από το υπάρχον, ενώ η ανάθεση αντιγραφής αντικαθιστά το περιεχόμενο ενός ήδη υπάρχοντος αντικειμένου με το περιεχόμενο ενός άλλου
- Τι θα πρέπει να επιστρέφει ο `operator=`;
  - Θα πρέπει να επιστρέφει αναφορά στο ίδιο το αντικείμενο *this έτσι ώστε να υποστηρίζονται αναθέσεις στη σειρά όπως: a = b = c;

Αναφορές

- http://www.cs.fsu.edu/~xyuan/cop3330/