Προγραμματισμός Υπολογιστών με C++

ΚΑΤΑΣΚΕΥΑΣΤΕΣ ΑΝΤΙΓΡΑΦΗΣ

Γεώργιος Παπαϊωάννου (2013-16)
gepap@aueb.gr

Περιεχόμενο Παρουσίασης

- Περιγραφή:
  - Deep/Shallow copies
  - Η αναγκαιότητα των κατασκευαστών αντιγραφής
  - Ορισμός και χρήση κατασκευαστών αντιγραφής

- Τελευταία ενημέρωση: Νοέμβριος 2016

Deep / Shallow Copies

- Όσο οι κλάσεις μας γίνονται ολοένα και πιο πολύπλοκες, συχνά χρειάζεται να φυλάμε πεδία τύπου δείκτη σε δεδομένα που έχουν εκχωρηθεί στο σωρό ή συλλογές αντικειμένων
- Όταν δημιουργούμε ένα αντίγραφο ενός αντικειμένου, πρέπει να αντιγράψουμε όλα του τα πεδία στο νέο αντικείμενο
- Σε τι βάθος όμως προχωράμε την αντιγραφή;
- Ποιος είναι υπεύθυνος για την υλοποίηση αυτής της αντιγραφής; Μια εξωτερική συνάρτηση; Μια μέθοδος που προσδιορίζεται από εμάς;

Πότε Αντιγράφεται ένα Αντικείμενο;

- Όταν το περνάμε «κατά τιμή» σε μια συνάρτηση
- Όταν το αναθέτουμε σε μια μεταβλητή ως τιμή επιστροφής από μια συνάρτηση (εκτός αν είναι επιστροφή αναφοράς)
- Όταν καλούμε μια μη υπερφορτωμένη ανάθεση με τον τελεστή =
- Όταν δημιουργούμε ένα νέο αντικείμενο περνώντας στον κατασκευαστή του ένα υπάρχον ίδιου τύπου

… Δηλαδή εξαιρετικά συχνά!

Deep / Shallow Copies – Το Πρόβλημα (1)

```
class Array {
    float * buffer;
    int size;
public:
    Array(int items) {
        size = items; buffer = new float[items];
    }
    Array() : size(0), buffer(nullptr) {}
    ~Array() {
        if (buffer) delete[] buffer;
    }
    float& operator[](int index) {
        int i = index>size-1?size-1:index<0?0:index;
        return (buffer[i]);
    }
};
```

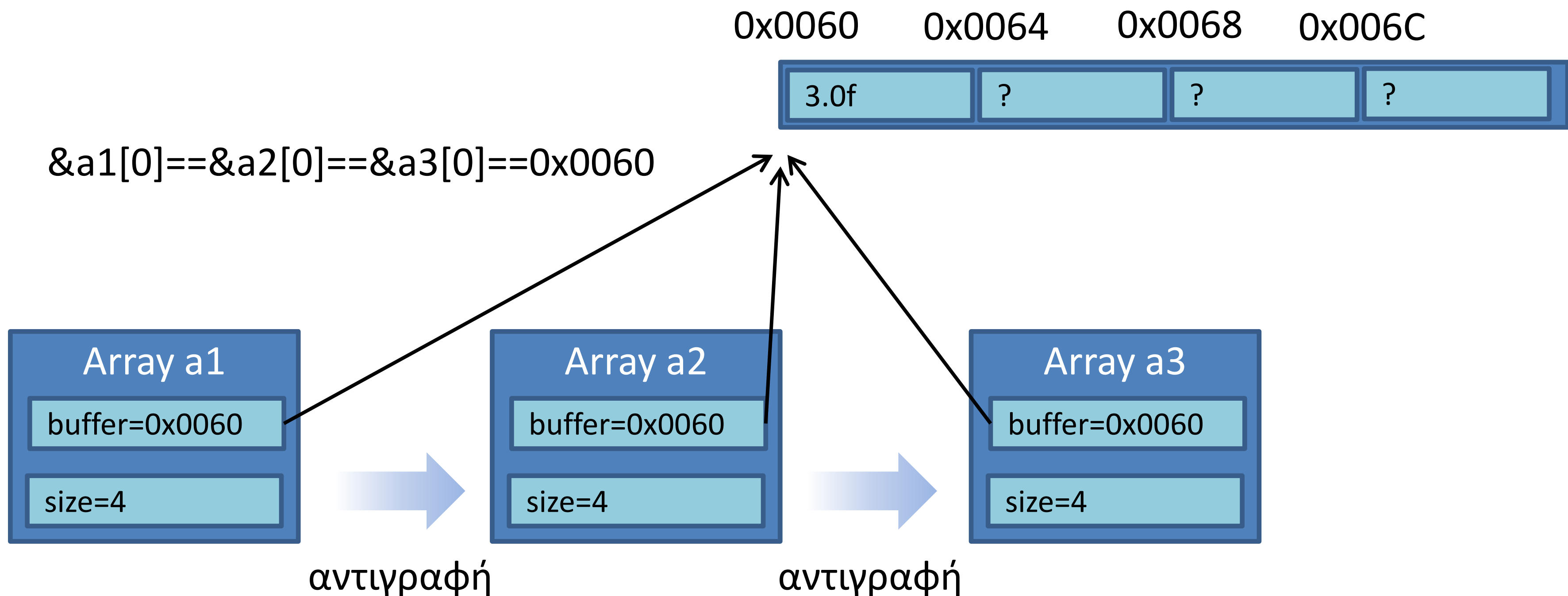
Deep / Shallow Copies – Το Πρόβλημα (2)

```
void main(int argc, char* argv[])
{
    Array a1(4);
    a1[0] = 1.f;

    Array a2 = a1;
    a2[0] = 2.f;

    Array a3(a2);
    a3[0] = 3.f;



    cout << a1[0] << ' ' << a2[0] << ' ' << a3[0] ;
    // Τυπώνει 3 3 3 αντί για 1 2 3 !!

}
```

Αντιγραφή του αντικειμένου

Deep / Shallow Copies – Το Πρόβλημα (3)

- Που βρίσκεται το πρόβλημα;
  - Κατά την αντιγραφή του αντικειμένου, αντιγράφονται όλα τα πεδία του, προφανώς κατά τιμή, έστω κι αν είναι δείκτες:



Deep / Shallow Copies – Το Πρόβλημα (4)

- Εδώ έγινε μια «ρηχή» αντιγραφή των πεδίων, ενώ στη συγκεκριμένη περίπτωση θα θέλαμε να είναι «βαθιά», δηλαδή να αντιγραφούν και τα περιεχόμενα των δεικτών ή αναφορών
- Αυτό δεν ισχύει πάντα και εξαρτάται από το συγκεκριμένο πρόβλημα που πάμε να λύσουμε:
  - Για παράδειγμα αν μια κλάση φυλάει ένα δείκτη σε μια άλλη κλάση που υλοποιεί ένα μοναδικό στιγμιότυπο ενός web service, δε θέλουμε κάθε φορά να παράγουμε και ένα πλήρες αντίγραφό του

Deep / Shallow Copies – Το Πρόβλημα (5)

- Στο παράδειγμά μας έχουμε και ένα δεύτερο πρόβλημα:
- Τι θα γίνει όταν κληθούν οι καταστροφείς των 3 αντικειμένων;
  - Ο πρώτος θα κληθεί επιτυχώς
  - Οι επόμενοι θα αποτύχουν γιατί θα προσπαθήσουν να αποδεσμεύσουν μνήμη που έχει ήδη αποδεσμευθεί!!

Ο Κατασκευαστής Αντιγραφής (Copy Constructor)

- Ποιος πραγματοποιεί αυτή την αντιγραφή;
- Ένας ειδικός κατασκευαστής, που ονομάζεται **κατασκευαστής αντιγραφής**
  - Αν δεν τον ορίσουμε, ο μεταγλωττιστής τον παράγει αυτόματα
  - Δημιουργεί ένα νέο αντικείμενο που να είναι αντίγραφο ενός άλλου, ίδιου τύπου
  - Το πρόβλημα είναι ότι το τί σημαίνει «αντίγραφο» δε μπορεί να το γνωρίζει ο μεταγλωττιστής πάντα (όπως στην περίπτωση που είδαμε)

Δικοί μας Κατασκευαστές Αντιγραφής

- Γι αυτό το λόγο, συχνά απαιτείται να ορίζουμε εμείς τον κατασκευαστή αντιγραφής μιας κλάσης ώστε να έχουμε πλήρη έλεγχο του τι συμβαίνει:
  - Προσδιορίζουμε για ποια πεδία χρειάζεται deep copy
  - Ο κατασκευαστής αντιγραφής, για μεγάλα αντικείμενα ενδέχεται να εκτελεί πολλές πράξεις και πιθανά να αντιγράφει πεδία που δε χρειάζεται:
    - βοηθητικά πεδία
    - πεδία που θα επαναπροσδιοριστούν έτσι κι αλλιώς στο νέο αντικείμενο μετά την κατασκευή
    - Μπορούμε να τα εξαιρέσουμε από την αντιγραφή, γλυτώνοντας πόρους

Παράδειγμα Κατασκευαστή Αντιγραφής (1)

```
class Array {
    float * buffer;
    int size;
public:
    Array(int items) {
        size = items; buffer = new float[items];
    }
    Array() : size(0), buffer(nullptr) {}
    ~Array() {
        if (buffer) delete[] buffer;
    }
    Array(const Array & proto) {
        size = proto.size;
        buffer = new float[size];
        memcpy(buffer, proto.buffer, size*sizeof(float));
    }

    ...
};
```

Συνάρτηση C αντιγραφής μπλοκ μνήμης. Χρήσιμη για γρήγορη αντιγραφή συνεχούς περιοχής δεδομένων

Παράδειγμα Κατασκευαστή Αντιγραφής (2)

```
void main(int argc, char* argv[])
{
    Array a1(4);
    a1[0] = 1.f;

    Array a2 = a1;
    a2[0] = 2.f;

    Array a3(a2);
    a3[0] = 3.f;


    cout << a1[0] << ' ' << a2[0] << ' ' << a3[0] ;
    // Τυπώνει 1 2 3
}
```

Καλείται ο δικός μας κατασκευαστής αντιγραφής: Κάθε αντικείμενο έχει το δικό του αντίγραφο του πίνακα buffer αντί για να δείχνει στην ίδια περιοχή μνήμης.

Δε δημιουργείται πλέον πρόβλημα με τη delete[] μέσα στον καταστροφέα

Παρατήρηση – Πρόσβαση σε Μη Δημόσια Πεδία ;!;

- Στον κώδικα που είδαμε, η μέθοδος του κατασκευαστή αντιγραφής είχε πρόσβαση σε προστατευμένα πεδία του αντικειμένου που περάσαμε. Γιατί μπορεί να το κάνει αυτό?!?!?
- Γιατί **ο προσδιοριστής ορατότητας των πεδίων εφαρμόζεται σε επίπεδο κλάσης, όχι αντικειμένου**!
- Δύο αντικείμενα της ίδιας κλάσης μπορούν να έχουν πρόσβαση το ένα στα μη δημόσια πεδία του άλλου

Ο Τελεστής Ανάθεσης με Αντιγραφή (=)

- Ο τελεστής ανάθεσης (=) επιχειρεί να δώσει την τιμή ενός δεδομένου σε ένα άλλο
- Όταν και οι 2 τελεσταίοι είναι του ίδιου τύπου, τότε αυτός ο τελεστής αποτελεί μια από τις ειδικές μεθόδους μιας κλάσεις, τον **copy assignment operator** (τελεστής ανάθεσης με αντιγραφή)
- Όπως με τον κατασκευαστή αντιγραφής, έτσι και στην περίπτωση του =, ο μεταγλωττιστής δημιουργεί έναν έτοιμο τελεστή =, αν δε δώσουμε το δικό μας

Πότε Καλείται ο Copy Assignment Operator;

- Καλείται όταν έχει δημιουργηθεί ένα αντικείμενο και γίνεται σε αυτό ανάθεση άλλου αντικειμένου ίδιου τύπου:

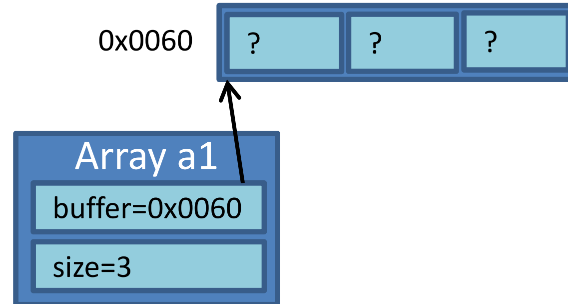
```
void main(int argc, char* argv[])
{
    Array a1(4);
    Array a2 = a1;
    a2 = Array(6);
}
```

Εδώ καλείται ο copy constructor, παρά το "=", αφού το a2 δεν έχει αρχικοποιηθεί ακόμα

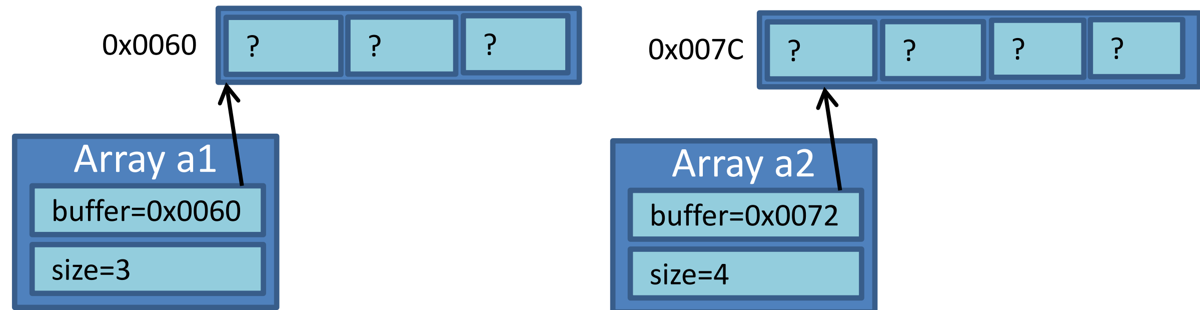
Εδώ καλείται ο copy assignment operator, μιας και το αντικείμενο a2 ήδη υπάρχει.

Default Copy Assignment – Τα Προβλήματα (1)

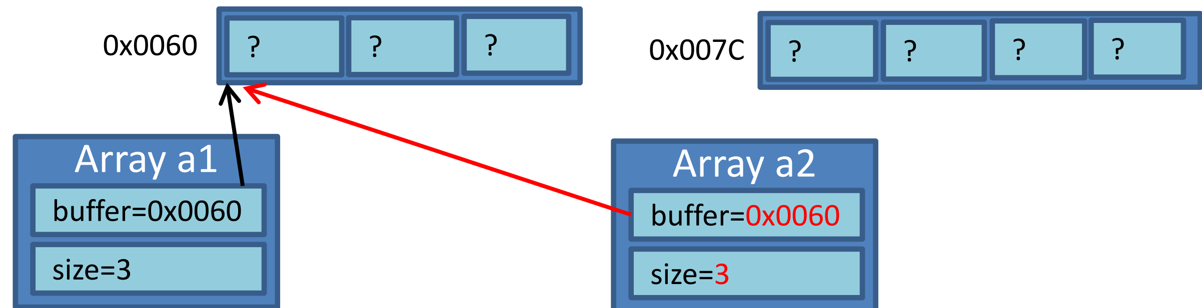
```
Array a1(3);
```



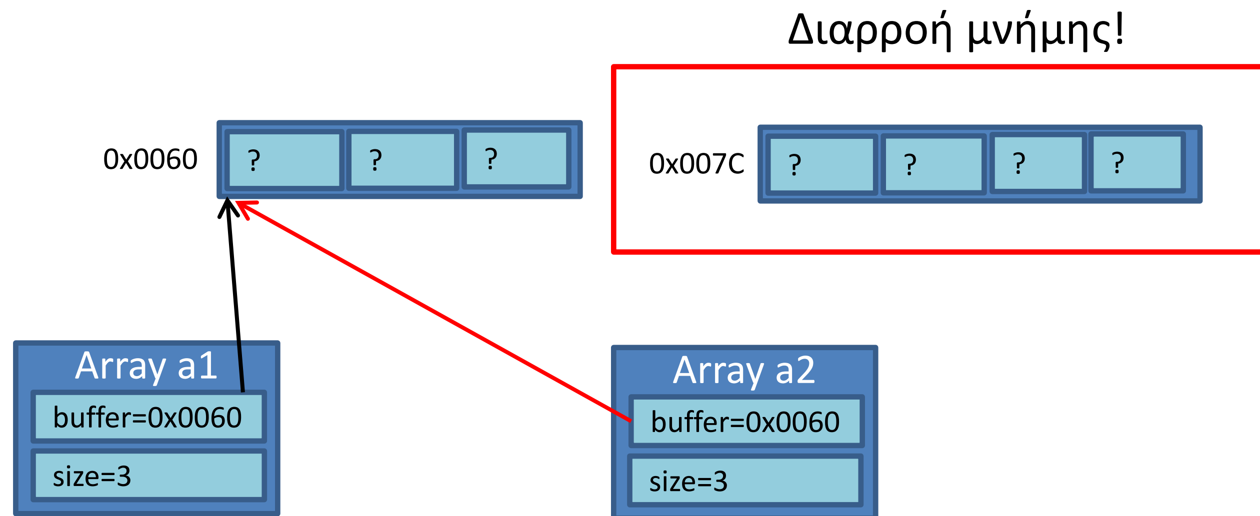

```
Array a2;
```




```
a2 = a1;
```



Default Copy Assignment – Τα Προβλήματα (2)




```
a1::~Array() {
    if (buffer) delete[] buffer;
}
```

```
a2::~Array() {
    if (buffer) delete[] buffer;
}
```

Υπερφόρτωση του τελεστή = (1)

```
class Array {
    ...

    Array(const Array & proto)  {
        size = proto.size;
        buffer = new float[size];
        memcpy(buffer,proto.buffer,size*sizeof(float));
    }

    void operator =(const Array & src) {
        if (buffer!=nullptr)
            delete[] buffer;
        size = src.size;
        buffer = new float[size];
        memcpy(buffer, src.buffer, size*sizeof(float));
     }
}
```

Υπερφόρτωση του τελεστή = (2)

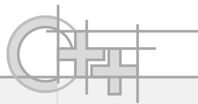
Ή καλύτερα:

```
class Array {
    ...
 private:
    void Replicate( const Array & in) {
        size = proto.size;
        buffer = new float[size];
        memcpy(buffer,proto.buffer,size*sizeof(float));
    }
 public:
    Array(const Array & proto)  {
        Replicate(proto);
    }

    void operator =(const Array & src) {
        if (buffer!=nullptr)
            delete[] buffer;
        Replicate(src);
    }
}
```

Υπερφόρτωση του τελεστή = (3)

```
void main(int argc, char* argv[])
{
    Array a1(4);
    Array a2, a3;
    a3 = a2 = a1;
}
```

Δε μπορούμε με την τρέχουσα υλοποίηση να κάνουμε αλυσιδωτή ανάθεση: Ο τελεστής δεν επιστρέφει τιμή.



Υπερφόρτωση του τελεστή = (4)

```
class Array {
    ...
    Array& operator =(const Array & src) {
        if (buffer!=nullptr)
            delete[] buffer;
        Replicate(src);
        return *this;
    }
}
```

Υπερφόρτωση του τελεστή = (5)

```
void main(int argc, char* argv[])
{
    Array a1(4);
    Array a2, a3;
    a3 = a2 = a1; // ok!
    a1 = a1;
}
```

Δεν απαγορεύεται, αλλά ούτε δουλεύει!
Καταστρέφονται τα περιεχόμενα του buffer

Υπερφόρτωση του τελεστή = (5)

```
class Array {
    ...
    Array& operator =(const Array & src) {
        if (&src==this)
            return *this;
        if (buffer!=nullptr)
            delete[] buffer;
        size = src.size;
        buffer = new float[size];
        memcpy(buffer, src.buffer, size*sizeof(float));
        return *this;
    }
}
```

Συγκρίνουμε τις διευθύνσεις των 2 αντικειμένων. Αν είναι ίδιες, πρόκειται για αυτό-ανάθεση, οπότε δεν κάνουμε τίποτα