

Τεχνολογίες Υλοποίησης Αλγορίθμων

Εισαγωγή στην C++ - 3

Χρήστος Δ. Ζαρολιάγκης

Καθηγητής

email: `zaro@ceid.upatras.gr`

Τμήμα Μηχ/κών Υπολογιστών & Πληροφορικής

Πανεπιστήμιο Πατρών

Εισαγωγή στην C++ - 3

- Έννοιες Αντικειμενοστρεφούς Προγραμματισμού
- Κλάσεις & Σχεδιασμός Βασισμένος σε Αντικείμενα (Classes & Object-Based Design)
- Κληρονομικότητα & Αντικειμενοστρεφής Σχεδιασμός (Inheritance & Object-Oriented Design)
- Εικονικές Συναρτήσεις (Virtual Functions)
- Φίλες Συναρτήσεις και Κλάσεις (Friends)
- Γενικευμένος Σχεδιασμός χρησιμοποιώντας Αρχέτυπα (Generic Design using Templates)

Αντικειμενοστρεφής Προγραμματισμός

Αντικειμενοστρεφής vs. Δομημένος/Διαδικασιακός Προγραμματισμός

- **Ομοιότητα:** ανάλυση ενός σύνθετου συστήματος (μεγάλο έργο) σε απλούστερα υποσυστήματα (υποέργα).
- **Θεμελιώδης διαφορά:** **Πώς** γίνεται αυτή η ανάλυση.

Διαδικασιακή ανάλυση:

- πρώτα το έργο αναλύεται σε αρθρώματα (modules)
- κατόπιν θα αναζητηθούν οι απαιτούμενες δομές δεδομένων

Αντικειμενοστρεφής ανάλυση:

- αντιμετωπίζει το λογισμικό σαν ένα σύνολο καλώς ορισμένων αντικειμένων τα οποία μοντελοποιούν τις οντότητες του έργου
- αυτά τα αντικείμενα αλληλεπιδρούν μεταξύ τους για να αποτελέσουν ένα σύστημα λογισμικού
- η διαδικασιακή ανάλυση θα αντιμετωπισθεί αφού το έργο έχει αναλυθεί σε αντικείμενα

- **Αντικείμενο:**

- Οντότητα η οποία έχει δομή και τοπική κατάσταση.
- Ορίζει λειτουργίες που μπορούν να προσπελάσουν ή να αλλάξουν αυτή την κατάσταση.

- **Αντικειμενοστρεφής προγραμματισμός (ΑΠ):**

είναι μια μέθοδος υλοποίησης στην οποία

- Τα αντικείμενα είναι οι θεμελιώδεις δομικές μονάδες.
- Κάθε αντικείμενο είναι το στιγμιότυπο ενός τύπου (ή κλάσης).
- Οι κλάσεις σχετίζονται μεταξύ τους μέσω σχέσεων *κληρονομικότητας*.

- **Αντικειμενοστρεφής γλώσσα:**

- (1) Υποστηρίζει αντικείμενα.
- (2) Απαιτεί τα αντικείμενα να ανήκουν σε κλάσεις (άμεσα ή έμμεσα).
- (3) Υποστηρίζει την κληρονομικότητα.

- **Ενθυλάκωση Δεδομένων (Data Encapsulation) ή Απόκρυψη Πληροφορίας (Information hiding):**

απόκρυψη των λεπτομερειών υλοποίησης ενός αντικειμένου δεδομένων από τον `έξω κόσμο`.

- **Αφαίρεση Δεδομένων (Data Abstraction):**

διαχωρισμός μεταξύ των *προδιαγραφών* ενός αντικειμένου δεδομένων και της *υλοποίησής* του.

- **Τύπος Δεδομένων (Data Type):**

Συλλογή αντικειμένων μαζί με ένα σύνολο λειτουργιών πάνω σε αυτά τα αντικείμενα.

- **Αφηρημένος Τύπος Δεδομένων – ΑΤΔ (Abstract Data Type):**

Τύπος Δεδομένων οργανωμένος κατά τέτοιο τρόπο έτσι ώστε οι προδιαγραφές των αντικειμένων και οι προδιαγραφές των λειτουργιών των αντικειμένων να απομονώνονται από την αναπαράσταση των αντικειμένων και την υλοποίηση των λειτουργιών.

● Κληρονομικότητα (Inheritance):

- Μηχανισμός που επιτρέπει την επέκταση της λειτουργικότητας ενός αντικειμένου.
- Επιτρέπει τη δημιουργία νέων τύπων με περιορισμένες ή πιο εκτενείς ιδιότητες σε σχέση με τον αρχικό τύπο.
- Ο αρχικός τύπος ονομάζεται τύπος βάσης (base type/object/class) ενώ ο νέος ονομάζεται παράγωγος (derived).

● Πολυμορφισμός:

- Η δυνατότητα χειρισμού διαφορετικών οντοτήτων με ομοιόμορφο τρόπο.
- Στην περίπτωση του αντικειμενοστρεφούς προγραμματισμού, η δυνατότητα αντικειμένων διαφορετικών κλάσεων (που έχουν πιθανά μεταξύ τους σχέσεις κληρονομικότητας) να αντιδρούν διαφορετικά στην ίδια λειτουργία.
- Όταν κάποιες λειτουργίες εφαρμόζονται σε έναν πολυμορφικό τύπο, εκτελείται αυτόματα εκείνη η λειτουργία η οποία είναι η κατάλληλη για τον συγκεκριμένο τύπο.

Στόχος Α-Π που επιτυγχάνεται:

επαναχρησιμοποίηση λογισμικού (software reusability).

Κλάσεις

Κλάση: Μηχανισμός της C++ για τον ορισμό νέων τύπων δεδομένων. Υποστηρίζει αφαίρεση δεδομένων, ενθυλάκωση δεδομένων και κληρονομικότητα.

Παράδειγμα μελέτης: υλοποίηση ενός ΑΤΔ για ακέραιους πίνακες σε δύο φάσεις:

(α) Σχεδιασμός βασισμένος σε αντικείμενα.

(β) Αντικειμενοστρεφής Σχεδιασμός.

Σχεδιασμός βασισμένος σε αντικείμενα

Ένα αρχικό σύνολο υποστηριζόμενων λειτουργιών:

1. Πραγματικό μέγεθος πίνακα.
2. Καταχώρηση (ανάθεση) ενός πίνακα σε άλλον και σύγκριση πινάκων.
3. Ερωτήσεις που αφορούν συγκεκριμένες τιμές στοιχείων: ελάχιστο, μέγιστο, εύρεση.
4. Δυνατότητα ταξινόμησης των στοιχείων του πίνακα.

Επίσης, πρέπει να υποστηρίζονται και οι λειτουργίες του συνήθη τύπου πίνακα:

5. Αρχικός προσδιορισμός μεγέθους του πίνακα (από τον χρήστη).
6. Αρχικοποίηση του πίνακα με ένα σύνολο τιμών.
7. Αριθμοδείκτης.
8. Έλεγχος ορθότητας του αριθμοδείκτη.

Γενική μορφή κλάσης που υποστηρίζει σχεδιασμό βασισμένο σε αντικείμενα:

```
class IntArray
{
    public:
        // public methods (member functions) and fields
        // comprises the public interface of the class
        // and can be accessed from any other function
        // of the program

    private:
        // private methods (member functions) and fields
        // can be accessed only by the member
        // functions of the class
        // private implementation and object state storage
};
```

Χρήση κλάσης για ορισμό αντικειμένων:

Οι τύποι/αντικείμενα που ορίζονται από τον προγραμματιστή χρησιμοποιούνται όπως και οι ενσωματωμένοι τύποι:

```
// a single IntArray class object
IntArray myarray;

// a pointer to a single IntArray class object
IntArray * pmyarray = new IntArray;
```

Ένα αρχικό σύνολο μεθόδων:

```
class IntArray
{
    public:

        int size() const; // member function which does
                          // not modify the class object

        IntArray& operator=(const IntArray &);
        bool operator==(const IntArray &) const;
        bool operator!=(const IntArray &) const;

        int min() const;
        int max() const;
        int find(int) const;

        void sort();

    private:
        // the private implementation
};
```

Κλήση μιας συνάρτησης μέλους

- Τελεστής προσπέλασης `.` (τελεία): χρησιμοποιείται από αντικείμενα της κλάσης.

```
IntArray myarray;  
int min_val = myarray.min();
```

- Τελεστής προσπέλασης `->` ("βέλος"): χρησιμοποιείται από δείκτες σε αντικείμενα της κλάσης.

```
IntArray *pmyarray = &myarray;  
int max_val = pmyarray->max();
```

Εφαρμογή Τελεστών

Γίνεται με τον ίδιο τρόπο όπως και στους ενσωματωμένους τύπους.

```
IntArray myarray1, myarray2;  
.  
.  
.  
myarray1 = myarray2;  
// The compiler translates this to  
// myarray1.operator=(myarray2);  
.  
.  
.  
if (myarray1 == myarray2)  
// The compiler translates this to  
// myarray1.operator==(myarray2)  
{ // do something  
}
```

Ποια είναι τα μέλη δεδομένων (data members) που είναι αναγκαία για την αναπαράσταση της κλάσης `IntArray`;

- Το μέγεθος του πίνακα όπως προσδιορίζεται από τον χρήστη.
- Κατανομή και αποθήκευση του πίνακα \Rightarrow μέλος δεδομένων τύπου δείκτη για αποθήκευση της διεύθυνσης που επιστρέφει ο τελεστής `new`.

```
class IntArray
{
    public:
        // ...
        int size() const { return asize; }

    private:
        int    asize;
        int *  ia;
};
```

Παρατήρηση: Μια συνάρτηση μέλους που ορίζεται μέσα σε ορισμό κλάσης θεωρείται αυτόματα ως `inline`.

Πώς αρχικοποιούνται τα αντικείμενα μιας κλάσης·

Η C++ παρέχει έναν αυτόματο μηχανισμό για κλάσεις ορισμένες από τον χρήστη.

Συνάρτηση κατασκευής (constructor):

- Ειδική συνάρτηση-μέλος που χρησιμοποιείται για αρχικοποίηση.
- Ορίζεται από τον προγραμματιστή.
- Έχει το ίδιο όνομα με αυτό της κλάσης.
- Δεν έχει τύπο επιστροφής τιμής.
- Μπορεί να ορισθεί πολλές φορές (μέσω υπερφόρτωσης συναρτήσεων).

```
class IntArray
{
    public:
        IntArray(int sz = DefaultArraySize);
        IntArray(const int * ar, int sz);
        IntArray(const IntArray & other);
        // ...

    private:
        static const int DefaultArraySize = 12;
        // ...
};
```

Η προεπιλεγμένη συνάρτηση κατασκευής:

```
                IntArray(int sz = DefaultArraySize);  
// initialises an object of size DefaultArraySize  
IntArray myarray1;  
  
// passes the argument 1000 to the constructor  
IntArray myarray2(1000);
```

Μια αρχική υλοποίηση της προεπιλεγμένης συνάρτησης κατασκευής

```
// :: class scope operator  
IntArray::IntArray(int sz)  
{  
    // set the data members  
    asize = sz;  
    ia = new int[asize];  
  
    // initialise the memory  
    for (int i=0; i < asize; ++i)  
        ia[i] = 0;  
}
```

Η δεύτερη συνάρτηση κατασκευής:

```
                IntArray(const int * array, int sz);  
  
// Initialises a new IntArray class object  
// with a built-in integer array  
  
int ia[10] = {0,1,2,3,4,5,6,7,8,9};  
IntArray myarray3(ia,10);
```

Υλοποίηση:

```
IntArray::IntArray(int * array, int sz)  
{  
    // set the data members  
    asize = sz;  
    ia = new int[asize];  
  
    // copy the data members  
    for (int i=0; ix < asize; ++i)  
        ia[ix] = array[ix];  
}
```

Η συνάρτηση κατασκευής αντιγράφου (copy constructor):

```
IntArray(const IntArray & iA);
```

Διαχειρίζεται την αρχικοποίηση ενός αντικειμένου της κλάσης `IntArray` με κάποιο άλλο. Ο `copy constructor` καλείται αυτόματα:

1. Όταν δηλώνεται μία μεταβλητή και αρχικοποιείται με την τιμή μιας άλλης.
2. Κατά τη μεταβίβαση ορισμάτων σε συνάρτηση κατ' αξία.
3. Κατά την επιστροφή τιμής από συνάρτηση κατ' αξία.

```
IntArray myarray4;
```

```
// equivalent initialisations  
IntArray A = myarray4;  
IntArray B(myarray4);
```

Υλοποίηση:

```
// copy constructor  
IntArray::IntArray(const IntArray & other)  
{  
    asize = other.asize;  
    ia = new int[asize];  
  
    for (int i=0; i < asize; ++i)  
        ia[i] = other.ia[i];  
}
```

Παρατήρηση: Αν δεν ορίζεται, τότε εκτελείται η προεπιλεγμένη αρχικοποίηση των μελών δεδομένων (default memberwise initialization).

Και οι τρεις συναρτήσεις κατασκευής υλοποιούνται με *παρόμοιο τρόπο*. Υπάρχει τρόπος να αποφύγουμε την μη επιθυμητή αναπαραγωγή του ίδιου κώδικα;

```
class IntArray
{
    public:
        // ...
    private:
        void init(const int * array, int sz);
        // ...
};

void IntArray::init(const int * array, int sz)
{
    asize = (sz < 1) ? 1 : sz;
    ia = new int[asize];

    for (int i=0; i < asize; ++i)
    {
        ia[i] = (!array) ? 0 : array[i];
    }
}
```

Οι τρεις συναρτήσεις κατασκευής μπορούν να ξαναγραφούν ως εξής:

```
IntArray::IntArray(int sz = DefaultArraySize)
    { init(0, sz); }
IntArray::IntArray(const int * ar, int sz)
    { init(ar, sz); }
IntArray::IntArray(const IntArray & other)
    { init(other.ia, other.asize); }
```

Συνάρτηση Κατάργησης (destructor):

- Ειδική συνάρτηση-μέλος που καλείται αυτόματα όταν τελειώνει η εμφάνιση του αντικειμένου στο πρόγραμμα.
- Ορίζεται από τον χρήστη.
- Έχει σαν όνομα το σύμβολο `~` ακολουθούμενο από το όνομα της κλάσης.
- Δεν έχει τύπο επιστροφής τιμής.

```
class IntArray
{
public:
    // constructors
    IntArray(int sz = DefaultArraySize) { init(0, sz); }
    IntArray(const int * ar, int sz)    { init(ar, sz); }
    IntArray(const IntArray & other)
        { init(other.ia, other.ysize); }

    // destructor
    ~IntArray() { delete[] ia; }
    // ...
private:
    void init(const int * array, int sz);
    // ...
};
```

Αριθμοδείκτης

Υποστηρίζεται με την παροχή μιας εξειδίκευσης του τελεστή αριθμοδείκτη ([]) σε σχέση με τη συγκεκριμένη κλάση.

Ο τελεστής επιστρέφει το συγκεκριμένο στοιχείο με αναφορά ώστε να είναι δυνατή η τροποποίησή του. Για να είναι δυνατή η κλήση του όμως και σε `const` στιγμιότυπα του `IntArray` απαιτείται και μία εκδοχή που θα επιστρέφει `const` αναφορά (μόνο για ανάγνωση συγκεκριμένου στοιχείου). Η C++ επιτρέπει την ύπαρξη δύο μεθόδων με ίδιο όνομα και ορίσματα αν η μία είναι `const`.

```
#include <cassert>
```

```
int& IntArray::operator[](int ix)
{
    // preprocessor macro to assert a precondition
    // if false, then a diagnostic message is printed
    // and the program terminates
    assert(ix>=0 && ix<size);
    return _ia[ix];
}
```

```
// version for const objects
const int& IntArray::operator[](int ix) const
{
    assert(ix>=0 && ix<size);
    return _ia[ix];
}
```

Καταχώρηση (ανάθεση τιμής) πίνακα: άλλη μια περίπτωση υπερφόρτωσης τελεστή

Ο τελεστής πρέπει να ελέγχει πιθανή ανάθεση του στιγμιότυπου στον εαυτό του και πρέπει να επιστρέφει το ίδιο το αντικείμενο για να επιτρέπονται αλυσιδωτές αναθέσεις.

```
IntArray& IntArray::operator=(const IntArray & other)
{
    if ( this == &other ) // check for assignment to self
        return *this;

    delete[] ia;
    init(other.ia, other.ysize);

    return *this; // returns the invoking object itself
                  // enables chain assignments
}
```

Ο έμμεσος δείκτης **this**

- Δείκτης που περιέχεται σε κάθε μέλος (συνάρτηση, τελεστή) κλάσης.
- Αναφέρεται στο αντικείμενο (το δεικτοδοτεί) για το οποίο καλείται η συνάρτηση-μέλος, ή ο τελεστής-μέλος.

Παράδειγμα

```
IntArray myarray1, myarray2;
```

```
// the returned object *this refers to myarray1
myarray1 = myarray2;
```

Αντικειμενοστρεφής Σχεδιασμός

Η μέχρι τώρα υλοποίηση του `IntArray`:

- (1) Υποστηρίζει έλεγχο των ορίων του αριθμοδείκτη.
- (2) Δεν υποστηρίζει ταξινομημένους πίνακες

Διαφορετικές κατηγορίες χρηστών μπορεί να έχουν αντιφατικές απαιτήσεις :

π.χ. μία κατηγορία να βρίσκει το (2) απολύτως απαραίτητο και ταυτόχρονα δεν θέλει την επιβάρυνση που συνεπάγεται το (1). Μια άλλη κατηγορία χρηστών πιστεύει ακριβώς το αντίθετο.

Πώς μπορεί η υλοποίησή μας να υποστηρίζει τις διαφορετικές απαιτήσεις διαφόρων χρηστών που θέλουν να την χρησιμοποιήσουν ;

Μια πρώτη προσπάθεια

Ανάπτυξη τριών διαφορετικών υλοποιήσεων της κλάσης, αντιγράφοντας το μεγαλύτερο μέρος του κώδικα και τροποποιώντας τον κατάλληλα για να υποστηρίζει ταξινομημένους πίνακες.

```
// Unsorted, without range-checking  
class IntArray { ... };
```

```
// Unsorted, with range-checking  
class IntArrayRC { ... };
```

```
// Sorted, without range-checking  
class IntSortedArray { ... };
```

Ποια είναι τα μειονεκτήματα αυτής της λύσης;

1. Διατήρηση τριών υλοποιήσεων πινάκων που περιέχουν ταυτόσημο το μεγαλύτερο μέρος του κώδικα.

Προτίμηση: ένα μοναδικό αντίγραφο του κοινού κώδικα να διαμοιράζεται από τις υπάρχουσες (και μελλοντικές) κλάσεις.

2. Συγγραφή ξεχωριστών συναρτήσεων για οποιαδήποτε λειτουργία πάνω σε αυτούς τους πίνακες. Π.χ.:

```
void process_array(IntArray &);  
void process_array(IntArrayRC &);  
void process_array(IntSortedArray &);
```

Προτίμηση: μια μοναδική συνάρτηση που να δέχεται σαν όρισμα τις υπάρχουσες (και μελλοντικές) κλάσεις.

Ο αντικειμενοστρεφής προγραμματισμός (στην C++) παρέχει τις εξής λύσεις:

1. *Κληρονομικότητα*: όταν μια νέα κλάση (`IntArrayRC`) κληρονομεί από μια άλλη, που ονομάζεται συνήθως *κλάση-βάσης* (`IntArray`), έχει προσπέλαση στα μέλη δεδομένων και στις συναρτήσεις-μέλη αυτής της κλάσης-βάσης χωρίς να απαιτείται αναπαραγωγή ταυτόσημου κώδικα. Η νέα κλάση χρειάζεται μόνο να παρέχει εκείνα τα μέλη δεδομένων και εκείνες τις συναρτήσεις-μέλη που είναι αναγκαίες για την υλοποίηση των επιπρόσθετων λειτουργιών που προσφέρει.
2. *Μηχανισμός Εικονικών Συναρτήσεων (Virtual functions)*: χειρίζεται κλήσεις συναρτήσεων-μελών που συμπεριφέρονται διαφορετικά, εξαρτώμενες από τον πραγματικό τύπο του αντικειμένου που καλείται στην ιεραρχία κλάσεων/υποκλάσεων (ή τύπων/υποτύπων).

Μία *υποκλάση (subclass)* έχει την ίδια διασύνδεση (interface) με την κλάση-βάσης, η οποία επιτρέπει στην κλάση-βάσης και στην υποκλάση να χρησιμοποιούνται εναλλακτικά μέσα σε ένα πρόγραμμα.

```
#include "IntArray.h"

void swap(IntArray & ia, int i, int j)
{
    int tmp = ia[i];
    ia[i] = ia[j];
    ia[j] = tmp;
}

...
IntArray IA;
IntArrayRC IARC;
IntArraySorted IAS;
string s("Non-IntArray");
...

swap(IA, 0, 10);    // ok, IA is of type IntArray
swap(IARC, 0, 10); // ok, IARC is a subtype of IntArray
swap(IAS, 0, 10);  // ok, IAS is a subtype of IntArray
swap(s, 0, 10);    // error, string is not a subtype
...                // of IntArray
```

Όμως,

- Ο τελεστής αριθμοδείκτη που καλείται από την `swap` πρέπει να αλλάζει δυναμικά σε κάθε κλήση και πρέπει να καθορίζεται από τον πραγματικό τύπο του πίνακα του οποίου τα στοιχεία ανταλλάσσονται.
- Αυτό επιτυγχάνεται με τη δήλωση αυτού του τελεστή ως **εικονικού** (*virtual*).

```
virtual int& operator[] (int ix) const;
```

- Το ίδιο μπορεί να γίνει και με κάθε άλλη συνάρτηση-μέλος η οποία είναι εξαρτώμενη από τύπους (type-dependent), π.χ. `min()`, `max()`, `find()`.

Πώς προετοιμάζουμε μια κλάση για κληρονομικότητα;

- *Οι συντακτικές αλλαγές είναι ελάχιστες:*
 - Μείωση του επιπέδου ενθυλάκωσης για να επιτρέπεται η προσπέλαση από τις παράγωγες κλάσεις της μη δημόσιας υλοποίησης.
 - Ρητός προσδιορισμός των συναρτήσεων που πρέπει να δηλωθούν ως εικονικές.
- *Σημαντική αλλαγή: σχεδιασμός της κλάσης-βάσης:*
 - Δημιουργία ενός προστατευμένου (`protected`) τμήματος για εκείνα τα μέλη δεδομένων και συναρτήσεις-μέλη τα οποία είναι ακόμη μη διαθέσιμα στο γενικό πρόγραμμα, αλλά είναι διαθέσιμα στις παράγωγες κλάσεις. Οτιδήποτε συμπεριληφθεί μέσα στο ιδιωτικό (`private`) τμήμα είναι διαθέσιμο μόνο στην κλάση-βάση και σε καμία παράγωγη κλάση.
 - Προσδιορισμός των συναρτήσεων-μελών που είναι εξαρτημένες από τύπους και δήλωσή τους ως εικονικές (`virtual`).

```

class IntArray
{
public:
    // constructors
    IntArray(int sz = DefaultArraySize) { init(0,sz); }
    IntArray(const int * ar, int sz)    { init(ar,sz); }
    IntArray(const IntArray & other) { init(other.ia,other.ysize); }

    // virtual destructor
    virtual ~IntArray() { delete[] ia; }

    IntArray& operator=(const IntArray&);
    int size() const { return ysize; }

    bool operator==(const IntArray&) const;
    bool operator!=(const IntArray&) const;

    // no range checking
    virtual int& operator[](int i) const {return ia[i];}

    virtual void sort(int, int);
    virtual int find(int) const;
    virtual int min() const;
    virtual int max() const;

protected:
    void init(const int*, int);
    static const int DefaultArraySize = 12;
    int    ysize;
    int * ia;
};

```

Τι πρέπει να κάνουμε για τις παράγωγες κλάσεις;

- Να τις δηλώσουμε ως παράγωγες από την κλάση-βάσης.
- Να παρέχουμε τα στιγμιότυπα των συναρτήσεων-μελών τους που έχουν δηλωθεί ως εικονικές στην κλάση-βάσης.
- Να παρέχουμε τις δικές τους συναρτήσεις κατασκευής (για όποια νέα μέλη προσθέτουν), διότι οι συναρτήσεις κατασκευής της κλάσης-βάσης δεν κληρονομούνται.

Οι συναρτήσεις κατασκευής καλούνται αυτόματα για όλη την ιεραρχία αρχίζοντας από την κλάση που βρίσκεται στη ρίζα του δένδρου κληρονομικότητας. Κάθε παράγωγη κλάση αρκεί επομένως να αρχικοποιεί μόνο τα πεδία που προσθέτει.

Οι συναρτήσεις κατάργησης καλούνται αυτόματα με την αντίστροφη σειρά.

Απαιτείται ένας τρόπος για να μεταβιβάζονται τα ορίσματα στις συναρτήσεις κατασκευής της κλάσης-βάσης αν αυτό είναι αναγκαίο.

Παράδειγμα: Πώς μεταβιβάζουμε το `IA` και το `7` στη συνάρτηση κατασκευής της κλάσης-βάσης `IntArray`:

```
int IA[] = {0, 12, 35, 2, 3, 12, 81};  
IntArrayRC IARC(IA, 7);
```

```

// IntArrayRC.h - part 1

#ifndef IntArrayRC_H
#define IntArrayRC_H

#include "IntArray.h"
#include <cassert>

class IntArrayRC : public IntArray
{
public:
    IntArrayRC(int sz = DefaultArraySize);
    IntArrayRC(const int *array, int array_size);
    IntArrayRC(const IntArrayRC &other);

    virtual int& operator[](int) const;

private:
    void check_range(int i) const
    {
        assert(i >= 0 && i < asize);
    }
};

```

```

// IntArrayRC.h - part 2

int& IntArrayRC::operator[](int index)
{
    check_range(index);
    return ia[index];
}

// portion after ':' is called member initialisation list
// it provides the mechanism by which the arguments are
// passed to the IntArray constructor

IntArrayRC::IntArrayRC(int sz)
    : IntArray(sz) {}

IntArrayRC::IntArrayRC(const int * iar, int sz)
    : IntArray(iar, sz) {}

IntArrayRC::IntArrayRC(const IntArrayRC &other)
    : IntArray(other.ia, other.asize) {}

// destructor is not provided, because the derived class
// doesn't introduce any data member requiring destruction

#endif

```

Είσοδος/Έξοδος

Η είσοδος/έξοδος μπορεί να γίνει με υπερφόρτωση των τελεστών << και >>. Επειδή το πρώτο όρισμα αυτών των τελεστών πρέπει να είναι το ρεύμα εισόδου ή εξόδου (λόγω του τρόπου κλήσης τους), δεν μπορούν να είναι μέθοδοι της κλάσης αλλά πρέπει να είναι εξωτερικές συναρτήσεις.

Πρέπει να επιστρέφουν το ρεύμα εισόδου/εξόδου ώστε να είναι δυνατή η εκτύπωση πολύπλοκων παραστάσεων.

```
ostream &operator<<(ostream &os, const IntArray &a)
{
    for (int i = 0; i << a.size(); ++i)
        os << a[i] << " ";
    return os;
}
...
IntArray IA;
cout << "Array: " << IA << "\n";
```

Φίλες Συναρτήσεις και Κλάσεις

Μερικές φορές μπορεί να θέλουμε να επιτρέψουμε την προσπέλαση των ιδιωτικών μελών (δεδομένα, συναρτήσεις) μιας κλάσης από άλλες κλάσεις ή συναρτήσεις. Αυτό μπορεί να επιτευχθεί δηλώνοντας αυτές τις κλάσεις ή συναρτήσεις ως *φίλες (friends)*.

```

class ClassOne
{
    friend ClassTwo;
    friend void set_something(ClassOne &, int &);
    // ...
private:
    int i;
    int& foo(const int &);
};

class ClassTwo;
{
    // ...
    ClassTwo(int & z) {ClassOne::foo(z)};
    // ...
};

void set_something(ClassOne & Obj, int & x)
{
    // ...
    Obj.i = defaultvalue;
    // ...
}

```

Γενικευμένος Σχεδιασμός με Αρχέτυπα

```
template <typename elemType>
class Array
{
public:
    Array(int sz = DefaultArraySize) { init(0, sz); }
    Array(const elemType *ar, int sz) { init(ar, sz); }
    Array(const Array & other) { init(other.ia, other.ysize); }

    virtual ~Array() { delete[] ia; }

    Array& operator=(const Array &);
    int size() const { return ysize; }

    virtual elemType& operator[](int i) const
        {return ia[i];}

    virtual void sort(int, int);
    virtual int find(const elemType &) const;
    virtual elemType min() const;
    virtual elemType max() const;

protected:
    void init(const elemType *, int);
    static const int DefaultArraySize = 12;
    int ysize;
    elemType * ia;
};
```

```

// ArrayRC.h

#ifndef ARRAYRC_H
#define ARRAYRC_H

#include "Array.h"

template <typename elemType>
class ArrayRC : public Array<elemType>
{
public:
    ArrayRC(int sz = DefaultArraySize)
        : Array<elemType>(sz) {}

    ArrayRC(const ArrayRC & r)
        : Array<elemType>(r) {}

    ArrayRC(const elemType *ar, int sz)
        : Array<elemType>(ar, sz) {}

    elemType& operator[](int i);
    const elemType& operator[](int i) const;
};

#endif

```

```

// ArrayRC.C

#include "ArrayRC.h"
#include "Array.C"
#include <cassert>

template <typename elemType>
elemType& ArrayRC<elemType>::operator[](int i)
{
    assert( i >= 0 && i < asize );
    return ia[i];
}

template <typename elemType>
const elemType& ArrayRC<elemType>::operator[](int i) const
{
    assert( i >= 0 && i < asize );
    return ia[i];
}

```