



ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ  
ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΪΚΑ ΜΑΘΗΜΑΤΑ



## ΥΠΟΛΟΓΙΣΤΕΣ II

Τάξεις

Διδάσκοντες: **Αν. Καθ. Δ. Παπαγεωργίου,**  
**Αν. Καθ. Ε. Λοιδωρικής**



Ευρωπαϊκή Ένωση  
Ευρωπαϊκό Κοινωνικό Ταμείο  
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ  
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ  
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

## Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



# ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ

## ΤΑΞΕΙΣ

## πέρα απο απλές δομές...

- Στο προηγούμενο κεφάλαιο γενικεύσαμε τις μεταβλητές
  - Δομές: σύνθετες μεταβλητές
  - Συναρτήσεις που δέχονται και επιστρέφουν δομές
- Τώρα γενικεύουμε μεταβλητές και συναρτήσεις
  - Τάξεις: αντικείμενα που περιέχουν
    - Μεταβλητές
    - Συναρτήσεις που επεξεργάζονται τις μεταβλητές
- Μια τάξη εισάγει ένα αντικείμενο με συγκεκριμένες ιδιότητες και λειτουργίες → αντικειμενοστρεφής προγραμματισμός

## Ορισμός τάξης

- Μια τάξη ορίζεται με παρόμοιο τρόπο που ορίζεται και η δομή
  - Χρησιμοποιείται η δεσμευμένη λέξη «class»
  - Μέλη της τάξης μπορεί να είναι μεταβλητές και συναρτήσεις
- Τα μέλη μιας τάξης χωρίζονται σε δύο είδη
  - «private»: ιδιωτικά μέλη: δεν είναι ορατά έξω από την τάξη
  - «public»: δημόσια μέλη: είναι ορατά από οποιοδήποτε σημείο του προγράμματος
- Εξ ορισμού η C++ θεωρεί ότι τα μέλη μιας τάξης είναι private, εκτός και εάν δηλωθούν public
  - Σε αντιδιαστολή, τα μέλη μιας δομής θεωρούνται εξ ορισμού «public»

## Ορισμός τάξης

- Ο γενικός ορισμός μιας τάξης είναι

```
class όνομα{
    τύπος μεταβλητή1 ;
    τύπος μεταβλητή2 ;
    τύπος όνομα-συνάρτησης1 (...);
    τύπος όνομα-συνάρτησης2 (...);
    .
    .
public:
    τύπος μεταβλητή3 ;
    τύπος μεταβλητή4 ;
    τύπος όνομα-συνάρτησης3 (...);
    τύπος όνομα-συνάρτησης4 (...);
    .
    .
};
```

Ιδιωτικά μέλη της τάξης. Δεν μπορούμε να τα προσπελάσουμε, παρά μόνο μέσω των δημόσιων συναρτήσεων της τάξης → ενθυλάκωση

Δημόσια μέλη της τάξης. Μπορούμε να τα προσπελάσουμε άμεσα από οποιοδήποτε σημείο του προγράμματος.

Οι δημόσιες συναρτήσεις επεξεργάζονται ιδιωτικά μέλη της τάξης 4

## Απλές τάξεις

- Στην πιο απλή περίπτωση, μια τάξη έχει
  - **Ιδιωτικές μεταβλητές:**
    - δεδομένα «προστατευμένα» από το υπόλοιπο πρόγραμμα
    - Επεξεργασία μόνο με προκαθορισμένο τρόπο
  - **Δημόσιες συναρτήσεις:**
    - Συναρτήσεις που επεξεργάζονται τις ιδιωτικές μεταβλητές
    - Συνιστούν την γέφυρα επικοινωνίας των μεταβλητών με το έξω πρόγραμμα

```
class όνομα{
    ιδιωτικές μεταβλητές;
public:
    δημόσιες συναρτήσεις;
};
```

5

## Σχέση δομής και τάξης

- Μια τάξη είναι μια γενίκευση της δομής. Άρα ότι μάθαμε με τις δομές, μπορούμε να το κάνουμε με τάξεις.
- Παράδειγμα, ένα «δημόσιο» διάνυσμα μπορεί να γραφτεί ισοδύναμα με δύο τρόπους

- Με δομή: 

```
struct vector { double x, y, z; };
```

- Με τάξη: 

```
class vector {
public:
    double x, y, z;
};
```

Τα παραπάνω είναι απολύτως ίδια. Τα παραδείγματα του προηγούμενου κεφαλαίου θα μπορούσαν να είχαν γραφτεί με την αντίστοιχη τάξη

## Συναρτήσεις-μέλη της τάξης

- Μέσα στην τάξη δηλώνουμε τα πρωτότυπα των συναρτήσεων-μελών
  - Παράδειγμα, ένα ιδιωτικό διάνυσμα, με μια συνάρτηση μέλος για την εισαγωγή τιμών στο διάνυσμα

```
class vector{
    double x, y, z;
public:
    void set( double x1, double y1, double z1) ;
};
```

- Τα **x**, **y**, **z** είναι ιδιωτικά, και δεν μπορούμε να τους δώσουμε τιμές κατευθείαν από το πρόγραμμα με τον τελεστή της τελείας.
- Θα χρησιμοποιήσουμε την συνάρτηση **set**, στην οποία δίνουμε στην λίστα εισόδου τις επιθυμητές τιμές
- Η **set** θα πρέπει να αντιγράψει τις τιμές στα **x**, **y**, **z**

7

## Δήλωση συνάρτησης μέλους

- Μια συνάρτηση που είναι μέλος μιας τάξης έχει άμεση πρόσβαση στα ιδιωτικά μέλη της τάξης
  - Δεν χρειάζεται τελείες κτλ
- Ως εκ τούτου, ανήκει στην τάξη και πρέπει να δηλωθεί ανάλογα
  - Ο μεταφραστής πρέπει να ξέρει την τάξη που ανήκει
- Δήλωση συνάρτησης (έξω από την τάξη, έξω από την main) :

```
τύπος τάξη::όνομα (λίστα εισόδου){
    εντολές;
    return ...;
}
```

Το καινούριο στοιχείο: πριν το όνομα της συνάρτησης γράφουμε το όνομα της τάξης, χωρισμένα με δύο άνω-κάτω τελείες

## Δήλωση συνάρτησης μέλους

- Στο προηγούμενο παράδειγμα του διανύσματος

```
class vector{
    double x, y, z;
public:
    void set( double x1, double y1, double z1) ;
};

void vector::set (double x1, double y1, double z1){
    x = x1;
    y = y1;
    z = z1;
}
```

- Η συνάρτηση έχει άμεση πρόσβαση στα  $x$ ,  $y$ ,  $z$ . Τα γνωρίζει εξ ορισμού επειδή ανήκουν στην ίδια τάξη

9

## Κλήση συνάρτησης μέλους

- Η κλήση γίνεται με τον τελεστή της τελείας, όπως γίνονταν και στις δομές.
- Μόνη διαφορά, ότι η συνάρτηση έχει πάντα παρενθέσεις
  - Με την λίστα εισόδου εάν υπάρχει
  - Άδειες παρενθέσεις εάν δεν υπάρχει εισόδος

- Παράδειγμα #1:

Πρόγραμμα που δηλώνει τάξη για διάνυσμα με δύο συναρτήσεις:

- μια για εισαγωγή τιμών
- μια για υπολογισμό του μέτρου του διανύσματος

10

## Παράδειγμα #1: διάνυσμα (1/2)

```
#include <iostream>
#include <cmath>
using namespace std;

class vector{
    double x, y, z;
public:
    void set(double, double, double);
    double magn();
};

void vector::set (double x1, double y1, double z1){
    x = x1;
    y = y1;
    z = z1;
}

double vector::magn(){
    return sqrt(x*x + y*y +z*z);
}
```

συνεχίζεται...

## Παράδειγμα #1: διάνυσμα (2/2)

```
int main(){
    vector v, u;
    double a, b, c;

    cout<<"εισάγετε συνιστώσες διανύσματος"<< endl;
    cin >> a >> b >> c;
    v.set(a, b, c);

    cout<<"εισάγετε συνιστώσες διανύσματος"<< endl;
    cin >> a >> b >> c;
    u.set(a, b, c);

    cout <<"το μέτρο του πρώτου διανύσματος"<<endl;
    cout << v.magn() << endl;

    cout <<"το μέτρο του δεύτερου διανύσματος"<<endl;
    cout << u.magn() << endl;
}
```

## Συνάρτηση δόμησης

- Κατά την δημιουργία ενός αντικειμένου, οι εσωτερικές μεταβλητές δεν έχουν αρχική τιμή
  - Στο προηγούμενο χρησιμοποιήσαμε την `set` για να δώσουμε αρχικές τιμές
  - η συνάρτηση `set` καλείται μέσα στο πρόγραμμα
- Μπορούμε να προγραμματίσουμε έτσι ώστε κατά την δήλωση του αντικειμένου, οι εσωτερικές μεταβλητές να παίρνουν εξ ορισμού κάποια συγκεκριμένη αρχική τιμή
- Συνάρτηση δόμησης**
  - ίδιο όνομα με αυτό της τάξης
  - χωρίς τύπο
  - με ή χωρίς λίστα εισόδου

13

## Παράδειγμα με συνάρτηση δόμησης

```
#include <iostream>
#include <cmath>
using namespace std;

class vector{
    double x, y, z;
public:
    vector();
    void set(double, double, double);
    double magn();
};

vector::vector(){ x = y = z = 0;}

void vector::set (double x1, double y1, double z1){
    x = x1; y = y1; z = z1; }

double vector::magn(){ return sqrt(x*x + y*y +z*z); }
```

## Παράδειγμα με συνάρτηση δόμησης

```
int main(){
    vector v, u;
    .
```

Στη δήλωση, τα διανύσματα `v`, `u` αρχικοποιούνται στο `(0,0,0)`.

- Η συνάρτηση δόμησης μπορεί και να έχει εισοδο

```
class vector{ double x, y, z;
public:
    vector(double x1, double y1, double z1);
    .
};
vector::vector(double x1, double y1, double z1){
    x = x1; y = y1; z = z1; }
```

- Τώρα όμως θα πρέπει να δοθούν οι τιμές κατά την δήλωση

```
int main(){
    vector v(0,0,0), u(1,1,1);
    .
```

Στη δήλωση, τα διανύσματα `v`, `u` αρχικοποιούνται στο `(0,0,0)` και `(1,1,1)`

## Παράδειγμα #2: λίστα αποθήκης

Πρόγραμμα για οργάνωση και χειρισμό αντικειμένων σε αποθήκη

- Θα ορίσουμε μια νέα τάξη που θα περιλαμβάνει
  - Μεταβλητές για κάθε αντικείμενο:
    - κόστος αγοράς
    - Τιμή πώλησης
    - Αριθμός αντικειμένων στην αποθήκη
    - Χρηματικό ισοζύγιο
  - Συναρτήσεις για επεξεργασία των παραπάνω
    - Συνάρτηση για αρχικές συνθήκες
    - Συνάρτηση για πώληση
    - Συνάρτηση για αγορά
    - Συνάρτηση που να δείχνει την κατάσταση

## Παράδειγμα #2: λίστα αποθήκης (1/5)

ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ - ΤΑΞΕΙΣ

```
#include <iostream>
using namespace std;

class item{
    double cost;
    double price;
    int stock;
    double cash;
public:
    item();
    void reset();
    void sell();
    void buy();
    void info();
};

item::item(){ cost = price = stock = cash = 0; }
```

## Παράδειγμα #2: λίστα αποθήκης (2/5)

ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ - ΤΑΞΕΙΣ

```
void item::sell(){

    int n;
    cout << "πόσα κομμάτια προς πώληση;" << endl;
    cin >> n;

    if ( stock<n ) cout<< "έχουμε μόνο" << stock <<endl;

    else {
        stock -= n;
        cash += n * price;
    }
}
```

18

## Παράδειγμα #2: λίστα αποθήκης (3/5)

ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ - ΤΑΞΕΙΣ

```
void item::buy(){

    int n;
    cout << "πόσα κομμάτια προς αγορά;" << endl;
    cin >> n;

    stock += n;
    cash -= n * cost;
}
```

19

## Παράδειγμα #2: λίστα αποθήκης (4/5)

ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ - ΤΑΞΕΙΣ

```
void item::reset(){
    cout << "παλιές τιμές αγοράς και πώλησης" << endl;
    cout << cost << " " << price << endl;
    cout << "εισάγετε νέες τιμές" << endl;
    cin >> cost >> price;
}

void item::info(){
    cout << stock <<" έχουν μείνει στην αποθήκη " <<endl;
    cout << cost <<" ευρώ η τιμή αγοράς " <<endl;
    cout << price <<" ευρώ η τιμή πώλησης " <<endl;
    cout << cash <<" ευρώ το ισοζύγιο " <<endl;
}
```

20

## Παράδειγμα #2: λίστα αποθήκης (5/5)

```
int main(){
    item a[1000]; int i, j;

    do {
        cout<< "Αριθμός προϊόντος;" <<endl; cin >> i;
        cout<<"Εντολή; 0-4: e-s-b-r-i"<<endl; cin >> j;

        switch (j){
            case 1: a[i].sell(); break;
            case 2: a[i].buy(); break;
            case 3: a[i].reset(); break;
            case 4: a[i].info(); break;
        }
    }
    while( j > 0 );
}
```

21

## Παράδειγμα #2: λίστα αποθήκης Εκτέλεση και αποτελέσματα

- |                                   |                               |
|-----------------------------------|-------------------------------|
| > Αριθμός προϊόντος;              | > Αριθμός προϊόντος;          |
| > 1                               | > 1                           |
| > Εντολή; 0-4: e-s-b-r-i          | > Εντολή; 0-4: e-s-b-r-i      |
| > 3                               | > 1                           |
| > παλιές τιμές αγοράς και πώλησης | > πόσα κομμάτια προς πώληση;  |
| > 0 0                             | > 3                           |
| > εισάγετε νέες τιμές             | > Αριθμός προϊόντος;          |
| > 3 4                             | > 1                           |
| > Αριθμός προϊόντος;              | > Εντολή; 0-4: e-s-b-r-i      |
| > 1                               | > 4                           |
| > Εντολή; 0-4: e-s-b-r-i          | > 2 έχουν μείνει στην αποθήκη |
| > 2                               | > 3 ευρώ η τιμή αγοράς        |
| > πόσα κομμάτια προς αγορά;       | > 4 ευρώ η τιμή πώλησης       |
| > 5                               | > -3 ευρώ το ισοζύγιο         |

22

## Παράδειγμα #2: λίστα προτεραιότητας

- Μια λίστα προτεραιότητας (ή λίστα αναμονής, ή «ουρά»)
  - Δέχεται νούμερα, που προστίθενται στο τέλος
  - Τραβάει νούμερα από την αρχή της ουράς
  - Η σειρά που ακολουθείται είναι «πρώτο μπαίνει, πρώτο βγαίνει)
- Από μεταβλητές θα χρειαστούμε
  - έναν πίνακα για να αποθηκεύουμε τα νούμερα
  - έναν ακέραιο για την θέση όπου βάζουμε τον επόμενο
  - έναν ακέραιο για την θέση απ' όπου τραβάμε τον επόμενο
- Από συναρτήσεις θα χρειαστούμε
  - μια συνάρτηση όταν ένας νέος αριθμός μπαίνει στη λίστα
  - μια συνάρτηση για να τραβάμε έναν αριθμό από την λίστα
    - και οι δύο πρέπει να ελέγχουν μην ξεπεραστούν όρια.
  - μια συνάρτηση να δίνει αρχικές τιμές.

23

## Παράδειγμα #2: λίστα προτεραιότητας (1/4)

- Θα ονομάσουμε την τάξη μας `queue` (ουρά)

```
class queue {
    double q[1000];
    int sloc, rloc;
public:
    queue();
    void qput(double);
    double qget();
};
```

- Η συνάρτηση δόμησης πρέπει να δίνει τις κατάλληλες αρχικές τιμές κατά την δήλωση μιας νέας ουράς.
  - Αρχικά είναι άδεια, άρα `sloc=rloc=0`;

```
queue::queue() { sloc = rloc = 0; }
```

24

## Παράδειγμα #2: Λίστα προτεραιότητας (2/4)

- Η συνάρτηση `qput` δέχεται έναν ρητό και τον προσθέτει στο τέλος της ουράς
  - Ελέγχει ότι η ουρά δεν έχει γεμίσει
  - Η θέση στο τέλος της ουράς είναι η `sloc`

```
void queue::qput(double a){
    if(sloc == 1000){
        cout<< "η λίστα είναι γεμάτη" << endl;
        return;
    }
    q[sloc] = a;
    sloc++;
}
```

25

## Παράδειγμα #2: Λίστα προτεραιότητας (3/4)

- Η συνάρτηση `qget` τραβάει έναν ακέραιο από την αρχή της ουράς
  - Ελέγχει ότι η ουρά δεν είναι άδεια
  - Η θέση στην αρχή της ουράς είναι η `rloc`

```
double queue::qget(){
    if(rloc == sloc){
        cout<< "η λίστα είναι άδεια ";
        return 0;
    }
    double r = q[rloc];
    rloc++;
    return r;
}
```

26

## Παράδειγμα #2: Λίστα προτεραιότητας (4/4)

- Ένα παράδειγμα προγράμματος που χρησιμοποιεί την τάξη `queue`

```
int main(){
    queue a, b;
    a.qput(10);
    b.qput(19);
    a.qput(20);
    b.qput(1);

    cout<<"η λίστα a περιέχει τους αριθμούς: ";
    cout << a.qget() << " " << a.qget() <<endl;

    cout<<"η λίστα b περιέχει τους αριθμούς: ";
    cout << b.qget() << " " << b.qget() << endl;
}
```

Στην εκτέλεση του παραπάνω θα λάβουμε

- > η λίστα a περιέχει τους αριθμούς: 10 20
- > η λίστα b περιέχει τους αριθμούς: 19 1

27

## Παράδειγμα #2: Λίστα προτεραιότητας 2

- Ένα δεύτερο παράδειγμα προγράμματος με την τάξη `queue`

```
int main(){
    queue a; int i; double x;
    do { cout<<"Επόμενη εντολή; 0-2 e-p-g"<<endl;
        cin >> i;

        switch (i) {
            case 1:
                cout<< "δώστε αριθμό" <<endl;
                cin >> x; a.qput(x); break;
            case 2:
                cout<< "επόμενος αριθμός στην ουρά ";
                cout << a.qget() << endl; break;
        }
    }
    while ( i > 0 );
}
```



## Παράδειγμα #2: Λίστα προτεραιότητας 2: εφαρμογή

```

> Επόμενη εντολή; 0-2 e-p-g
> 1
> δώστε αριθμό
> 4
> Επόμενη εντολή; 0-2 e-p-g
> 1
> δώστε αριθμό
> 7
> Επόμενη εντολή; 0-2 e-p-g
> 2
> επόμενος αριθμός στην ουρά 4
> Επόμενη εντολή; 0-2 e-p-g
> 1
> δώστε αριθμό
> 5
> Επόμενη εντολή; 0-2 e-p-g
> 2
> επόμενος αριθμός στην ουρά 7
> Επόμενη εντολή; 0-2 e-p-g
> 2
> επόμενος αριθμός στην ουρά 5
> Επόμενη εντολή; 0-2 e-p-g
> 2
> η λίστα είναι άδεια 0
> Επόμενη εντολή; 0-2 e-p-g
> 1
> δώστε αριθμό
> 15
> Επόμενη εντολή; 0-2 e-p-g
> 2
> επόμενος αριθμός στην ουρά 15

```

## Παράδειγμα #3: Μέτρηση καρτών

Πρόγραμμα που μετράει κάρτες σε παιχνίδι

- Έστω το παιχνίδι «21», το οποίο παίζεται ως εξής:
  - τραβάς όσα φύλλα επιθυμείς
  - ο πιο κοντά στο 21 κερδίζει
  - παίζουν τα φύλλα από 1 μέχρι και 10
  - παίζουν  $n$  τράπουλες

Θα φτιάξουμε μια τάξη που θα μετράει τα φύλλα

- Από μεταβλητές θα χρειαστούμε
  - έναν πίνακα για να αποθηκεύει πόσα φύλλα πέρασαν
  - έναν ακέραιο για πόσα φύλλα ακόμα παίζουν
- Από συναρτήσεις θα χρειαστούμε
  - μια συνάρτηση δόμησης για αρχικές τιμές
  - μια συνάρτηση όταν ένα νέο φύλλο περνάει
  - μια συνάρτηση που συμβουλεύει να τραβήξουμε ή όχι

## Παράδειγμα #3: Μέτρηση καρτών (1/4)

```

#include <iostream>
using namespace std;

class game{
    int cards[11];
    int num;
public:
    game(int);
    void add(int);
    double play(int);
};

game::game(int n){
    for (int i=1; i<=10; ++i) cards[i] = 4 * n;
    num = 40 * n;
}

```

## Παράδειγμα #3: Μέτρηση καρτών (2/4)

- Συνάρτηση για καταμέτρηση φύλλου που πέρασε
  - στην είσοδο το  $j$  είναι το φύλλο που περνάει
  - αφαιρείται ένα από τα  $j$  που παραμένουν
  - αφαιρείται ένα από τον συνολικό αριθμό που παραμένει

```

void game::add( int j ){

    if( cards[j] == 0 ){
        cout << "τέλειωσε αυτή η κάρτα" << endl;
        return;
    }

    cards[j]--;
    num--;
}

```

## Παράδειγμα #3: Μέτρηση καρτών (3/4)

- Συνάρτηση για εκτίμηση πιθανότητας να τραβήξουμε
  - στην είσοδο το `j` είναι το άθροισμα που έχουμε μέχρι τώρα
  - αθροίζουμε τον αριθμό φύλλων που δεν το καίνε
    - δηλαδή αυτά που έχουν μείνει από `1` μέχρι `21 - j`
  - διαιρούμε με τον συνολικό αριθμό φύλλων που έχει μείνει

```
double game::play( int j ){
    if( num == 0 ){
        cout<< "τελείωσαν οι κάρτες" << endl ;
        return 0;
    }

    double s=0;
    for(int i=1; i <= 21-j; ++i) s += cards[i];

    return s / num * 100;
}
```

## Παράδειγμα #3: Μέτρηση καρτών (4/4)

- Το κυρίως πρόγραμμα

```
int main(){
    int n, j;
    cout << "πόσες τράπουλες; ";
    cin >> n;
    game a(n);

    do{
        cout << "νέο φύλλο ";
        cin >> j;
        if (j <= 10) a.add(j);
        else cout << "τράβα" << a.play(j) << "%\n";
    }
    while( j > 0 );
}
```

## Υπερφόρτωση τελεστών

- Είδαμε πως γράφουμε συναρτήσεις για την εκτέλεση συγκεκριμένων λειτουργιών-πράξεων
- Είδαμε πως υπερφορτώνουμε συναρτήσεις
- Εδώ τα συνδυάζουμε υπερφορτώνοντας τους τελεστές
  - Για παράδειγμα, αντί να γράψουμε συνάρτηση `sum` για πρόσθεση διανυσμάτων, θα επαναπρογραμματίσουμε το `+` ώστε να κάνει την επιθυμητή λειτουργία
- Ως παράδειγμα, θα χρησιμοποιήσουμε την τάξη `vector`

## Πρόσθεση διανυσμάτων 1: με εξωτερική συνάρτηση

- Έστω η τάξη `vector`
  - για απλότητα θεωρούμε όλα τα μέλη της τάξης δημόσια

```
class vector{
public:
    double x, y, z;
};
```

- Θα γράψουμε εξωτερική συνάρτηση για την πρόσθεση

```
vector sum(vector v, vector u){
    vector w;
    w.x = v.x + u.x;
    w.y = v.y + u.y;
    w.z = v.z + u.z;
    return w;
}
```

## Πρόσθεση διανυσμάτων 1: με εξωτερική συνάρτηση

```
#include <iostream>
using namespace std;

class vector{
public: double x, y, z; };

vector sum(vector, vector);

int main(){
    vector v1, v2, v3;
    .
    .
    v3 = sum(v1, v2);
    .
    cout << "το άθροισμα v1+v2 είναι " << endl;
    cout << v3.x << v3.y << v3.z<<endl;
}
```

## Πρόσθεση διανυσμάτων 2: με συνάρτηση-μέλος

- Έστω η τάξη vector

```
class vector{
public:
    double x, y, z;
    vector sum(vector);
};
```

- Η συνάρτηση-μέλος δέχεται και επιστρέφει τον ίδιο τύπο με την τάξη
- καλείται από κάποια μεταβλητή τύπου `vector` με την τελεία

```
vector vector::sum(vector u) {
    vector w;
    w.x = x + u.x;
    w.y = y + u.y;
    w.z = z + u.z;
    return w;
}
```

38

## Πρόσθεση διανυσμάτων 2: με συνάρτηση-μέλος

```
#include <iostream>
using namespace std;

class vector{
public:
    double x, y, z;
    vector sum(vector);
};

int main(){
    vector v1, v2, v3;
    .
    .
    v3 = v1.sum(v2);
    .
    cout << "το άθροισμα v1+v2 είναι " << endl;
    cout << v3.x << v3.y << v3.z<<endl;
}
```

## Πρόσθεση διανυσμάτων 3: με υπερφόρτωση τελεστή

- Ορίζεται όπως και η εσωτερική συνάρτηση
- Για να δηλώσουμε ότι πρόκειται για τελεστή, το όνομά της αποτελείται από τον κωδικό `operator` συνοδευόμενο από το επιθυμητό σύμβολο
  - π.χ. για πρόσθεση, χρησιμοποιούμε το `operator+`

```
class vector{
public:
    double x, y, z;
    vector operator+(vector);
};
```

```
vector vector::operator+(vector u) {
    vector w;
    w.x = x + u.x;
    w.y = y + u.y;
    w.z = z + u.z;
    return w;
}
```

40

## Πρόσθεση διανυσμάτων 3: με υπερφόρτωση τελεστή

```
#include <iostream>
using namespace std;

class vector{
public:
    double x, y, z;
    vector operator+(vector);
};

int main(){
    vector v1, v2, v3;
    .
    .
    v3 = v1 + v2;
    .
    .
    cout << "το άθροισμα v1+v2 είναι " << endl;
    cout << v3.x << v3.y << v3.z<<endl;
}
```

## Σύγκριση συνάρτησης-μέλους και τελεστή

### Με συνάρτηση-μέλος

- Η συνάρτηση `sum` καλείται από το διάνυσμα στα αριστερά
  - Άρα έχει άμεση γνώση των μελών `x, y, z`
- Το διάνυσμα στα δεξιά εισέρχεται
  - Άρα για τα `x, y, z` του εισερχόμενου διανύσματος χρειαζόμαστε την τελεία

- Η συνάρτηση καλείται ως

```
v3 = v1.sum(v2);
```

### Με τελεστή

- Ο τελεστής `+` καλείται από το διάνυσμα στα αριστερά
  - Άρα έχει άμεση γνώση των μελών `x, y, z`
- Το διάνυσμα στα δεξιά εισέρχεται
  - Άρα για τα `x, y, z` του εισερχόμενου διανύσματος χρειαζόμαστε την τελεία

- Ο τελεστής καλείται ως

```
v3 = v1 + v2;
```

Με τελεστή δεν χρειάζονται η τελεία και οι παρενθέσεις!

42

## Υπερφόρτωση αφαίρεσης διανυσμάτων

```
vector vector::operator-(vector u){
    vector w;
    w.x = x - u.x;
    w.y = y - u.y;
    w.z = z - u.z;
    return w;
}
```

43

## Υπερφόρτωση γινομένου διανυσμάτων και γινομένου με ρητό

- Εσωτερικό γινόμενο δύο διανυσμάτων

```
double vector::operator*(vector u){
    return x*u.x + y*u.y + z*u.z;
}
```

- Γινόμενο διανύσματος με ρητό

```
vector vector::operator*(double m){
    vector w;
    w.x = x * m;
    w.y = y * m;
    w.z = z * m;
    return w;
}
```

44

## Δημιουργία τελεστή εξωτερικού γινομένου διανυσμάτων

- Εξωτερικό γινόμενο (ορίζουμε τον νέο τελεστή ^)

$$\vec{a} \times \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = \hat{i}(a_y b_z - a_z b_y) - \hat{j}(a_x b_z - a_z b_x) + \hat{k}(a_x b_y - a_y b_x)$$

```
vector vector::operator^(vector u) {
    vector w;
    w.x = y * u.z - z * u.y;
    w.y = z * u.x - x * u.z;
    w.z = x * u.y - y * u.x;
    return w;
}
```

45

## Λοιπές συναρτήσεις-μέλη

- Συνάρτηση δόμησης χωρίς είσοδο (για εξ ορισμού ανάθεση (0,0,0))

```
vector::vector() { x = y = z = 0; }
```

- Συνάρτηση δόμησης με είσοδο (για ανάθεση κατά την δήλωση)

```
vector::vector(double x1, double y1, double z1) {
    x = x1; y = y1; z = z1;
}
```

- Συνάρτηση set (για ανάθεση μετά την δήλωση)

```
void vector::set(double x1, double y1, double z1) {
    x = x1; y = y1; z = z1;
}
```

- Συνάρτηση magnitude

```
double vector::magnitude() {
    return sqrt(x*x + y*y + z*z);
}
```

## Παράδειγμα 1: Διανυσματικός λογισμός

- Πλήρες πρόγραμμα C++ με τους τελεστές και συναρτήσεις-μέλη που δόθηκαν προηγουμένως, ώστε να υπολογίσει την έκφραση

$$(\mathbf{v}_1 \times \mathbf{v}_2) \times (\mathbf{v}_1 \times \mathbf{v}_3) \cdot |(\mathbf{v}_1 + \mathbf{v}_2) \times (\mathbf{v}_1 - \mathbf{v}_3)|$$

47

## Παράδειγμα 1: Διανυσματικός λογισμός (1/2)

```
#include <iostream>
#include <cmath>
using namespace std;

class vector{
public:
    double x, y, z;
    vector();
    vector(double, double, double);
    void set(double, double, double);
    vector operator+(vector);
    vector operator-(vector);
    double operator*(vector);
    vector operator*(double);
    vector operator^(vector);
    double magnitude();
};
```

## Παράδειγμα 1: Διανυσματικός λογισμός (2/2)

```
int main(){
    double x, y, z; vector r;
    cout << "εισάγετε τα τρία διανύσματα " <<endl;

    cin >> x >> y >> z;
    vector v1(x,y,z);

    cin >> x >> y >> z;
    vector v2(x,y,z);

    cin >> x >> y >> z;
    vector v3(x,y,z);
    
$$(v_1 \times v_2) \times (v_1 \times v_3) \cdot |(v_1 + v_2) \times (v_1 - v_3)|$$

    r = ((v1^v2)^(v1^v3)) * ((v1+v2)^(v1-v3)).magn();

    cout << "το ζητούμενο διάνυσμα είναι" <<endl;
    cout << r.x <<" " << r.y <<" " << r.z <<endl;
}
```

## Παράδειγμα 2: μιγαδικοί αριθμοί

```
#include <iostream>
#include <cmath>
using namespace std;

class complex{
public:
    double real, imag;
    complex();
    complex(double, double);
    void set(double, double);
    complex operator+(complex);
    complex operator-(complex);
    complex operator*(complex);
    complex operator*(double);
    double magnitude();
};
```

## Παράδειγμα 2: μιγαδικοί αριθμοί

```
complex::complex(){real = imag = 0; }

complex::complex(double r, double i){
    real = r; imag = i;
}

void complex::set(double r, double i){
    real = r; imag = i;
}

double complex::magnitude(){
    return sqrt(real*real + imag*imag);
}
```

## Παράδειγμα 2: μιγαδικοί αριθμοί

```
complex complex::operator+(complex z){
    complex w;
    w.real = real + z.real;
    w.imag = imag + z.imag;
    return w;
}

complex complex::operator-(complex z){
    complex w;
    w.real = real - z.real;
    w.imag = imag - z.imag;
    return w;
}
```

## Παράδειγμα 2: μιγαδικοί αριθμοί

```

complex complex::operator*(complex z){
    complex w;
    w.real = real * z.real - imag * z.imag;
    w.imag = real * z.imag + imag * z.real;
    return w;
}

complex complex::operator*(double m){
    complex w;
    w.real = real * m;
    w.imag = imag * m;
    return w;
}

```

53

## Παράδειγμα 2: μιγαδικοί αριθμοί

Χρησιμοποιώντας την τάξη complex, υπολογίστε τους 100 πρώτους όρους της παρακάτω αναδρομικής ακολουθίας μιγαδικών αριθμών

$$z_n = rz_{n-1}(1 - z_{n-2})$$

$$z_1 = 0.75 + i0.25, z_2 = 0.25 + i0.75, r = 2$$

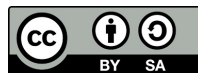
```

int main(){
    double r = 2;
    complex za(0.75, 0.25);
    complex zb(0.25, 0.75);
    complex z1(1.00, 0.00);

    for (int i=3; i<=100; ++i){
        zc = ( zb * (z1 - za) ) * r;
        za = zb; zb = zc;
        cout << "ο όρος " << i << "είναι "
             << zc.real << " " << zc.imag << endl;
    }
}

```

## Τέλος Ενότητας



## Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Ιωαννίνων**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «**Εκπαίδευση και Δια Βίου Μάθηση**» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Ευρωπαϊκή Ένωση  
Ευρωπαϊκό Κοινωνικό Ταμείο



ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ  
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ  
Περίοδος προγραμματισμού 2007-2013

ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ  
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



ΕΣΠΑ  
2007-2013  
Μίσθια για το μέλλον  
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ



## Σημειώματα

### Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Έχουν προηγηθεί οι κάτωθι εκδόσεις:

- Έκδοση 1.0 διαθέσιμη εδώ.

<http://ecourse.uoi.gr/course/view.php?id=1227>.

## Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Ιωαννίνων, Διδάσκοντες: Αν.  
Καθ. Δ. Παπαγεωργίου, Αν. Καθ. Ε. Λοιδωρίκης.  
«ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ. Τάξεις». Έκδοση: 1.0. Ιωάννινα 2014.  
Διαθέσιμο από τη δικτυακή διεύθυνση:  
<http://ecourse.uoi.gr/course/view.php?id=1227>.

## Σημείωμα Αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Παρόμοια Διανομή, Διεθνής Έκδοση 4.0 [1] ή μεταγενέστερη.



- [1] <https://creativecommons.org/licenses/by-sa/4.0/>.