

**Ανοικτά Ακαδημαϊκά Μαθήματα**

**Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας**

**Αντικειμενοστρεφής Προγραμματισμός (Ε)**

**Ενότητα 6:** Συναρτήσεις

Κλειώ Σγουροπούλου

Τμήμα Μηχανικών Πληροφορικής ΤΕ

|  |  |
| --- | --- |
| Το περιεχόμενο του μαθήματος διατίθεται με άδεια Creative Commons εκτός και αν αναφέρεται διαφορετικά | 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. |

Πίνακας Περιεχομένων

[Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 6 3](#_Toc402879636)

[Δείκτες σε παραγόμενες κλάσεις 3](#_Toc402879637)

[Παράδειγμα1 3](#_Toc402879638)

[Δυναμικές – Εικονικές (Virtual) Συναρτήσεις 4](#_Toc402879639)

[Παράδειγμα2 4](#_Toc402879640)

[Άσκηση 1 5](#_Toc402879641)

[Καθαρά Δυναμικές – Εικονικές (Virtual) Συναρτήσεις – Abstract κλάσεις 5](#_Toc402879642)

[Άσκηση 2 6](#_Toc402879643)

[Άσκηση 3 6](#_Toc402879644)

[Namespaces 6](#_Toc402879645)

[Static data και functions 7](#_Toc402879646)

[Παράρτημα - Παράδειγμα με window class hierarchy 8](#_Toc402879647)

# Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 6

## Δείκτες σε παραγόμενες κλάσεις

Οι δείκτες σε αντικείμενα μιας κλάσης και σε αντικείμενα κλάσεων που παράγονται από αυτήν σχετίζονται μεταξύ τους. Έτσι όταν ένας δείκτης διακηρυχτεί στον τύπο της βασικής κλάσης, μπορεί να χρησιμοποιηθεί για την προσπέλαση στοιχείων σε αντικείμενα των παραγόμενων κλάσεων.

### Παράδειγμα1

#include <iostream> using namespace std;

class Base{

int i;

public:

void set\_i(int m);

int out\_i(void);

};

class Child : public Base { int k;

public:

void set\_k(int m); int out\_k(void);

};

void Base::set\_i(int m){ i=m;

}

int Base::out\_i(void){ return i;

}

void Child::set\_k(int m){ k=m;

}

int Child::out\_k(void){ return k;

}

int main(){ Base \*pb; Base B\_obj; Child \*pd; Child D\_obj;

pb=&B\_obj;

pb->set\_i(3);

pb=&D\_obj; //Προσοχή: Δείκτης σε Base δείχνει σε αντικείμενο Child

pb->set\_i(4);

//pb->set\_k(5); //Δεν είναι εφικτό

cout << " i of B class = "<<B\_obj.out\_i()<<'\n'; cout <<" i of D class = "<<D\_obj.out\_i()<<'\n';

pd=&D\_obj;

pd->set\_k(5);

cout <<"k of D class = "<<D\_obj.out\_k()<<'\n'; return 0;

}

## Δυναμικές – Εικονικές (Virtual) Συναρτήσεις

Η C++ επιτυγχάνει τον πολυμορφισμό κατά την εκτέλεση ενός προγράμματος χρησιμοποιώντας παραγόμενες κλάσεις και δυναμικές συναρτήσεις, Οι δυναμικές συναρτήσεις διακηρύσσονται με τη λέξη κλειδί *virtual* στη βασική κλάση ενώ διαφορετικές εκδόσεις τους (χωρίς να είναι απαραίτητη η λέξη κλειδί *virtual*) διακηρύσσονται στις παραγόμενες κλάσεις. Οι τελευταίες μπορούμε να τις καλέσουμε χρησιμοποιώντας ένα δείκτη στον τύπο της βασικής κλάσης. Στην περίπτωση αυτή η C++ προσδιορίζει, κατά την εκτέλεση του προγράμματος, ποια συνάρτηση θα χρησιμοποιηθεί από τον τύπο του αντικειμένου του οποίου η διεύθυνση δόθηκε στο δείκτη.

Τα πρότυπα των δυναμικών συναρτήσεων πρέπει να ταυτίζονται κατά τη διακήρυξη τους τόσο στη βασική όσο και στις παραγόμενες κλάσεις, για να έχουμε τη δυνατότητα να συντάξουμε κώδικα γενικής μορφής τον οποίο να εκμεταλλεύονται διαφορετικού τύπου αντικείμενα. Επίσης οι δυναμικές συναρτήσεις πρέπει να είναι μέλη της βασικής και των παραγόμενων κλάσεων αλλά όχι φιλικές συναρτήσεις, χωρίς αυτό να τις αποκλείει να είναι φιλικές για άλλες κλάσεις.

### Παράδειγμα2

#include <iostream> using namespace std;

class Animal

{

public:

virtual void Display() {

cout << "Base Animal Display() \n"; }

};

class Cat : public Animal

{

public:

void Display(){

cout << "Derived Cat Display()\n";}

};

class Dog : public Animal

{

public:

void Display() {

cout << "Derived Dog Display()\n"; }

};

class Fish : public Animal

{

public:

//no Display() provided

};

int main()

{

Animal \*Animal\_ptr; Cat C;

Dog D; Fish F;

Animal\_ptr = &C; Animal\_ptr->Display();

Animal\_ptr = &D; Animal\_ptr->Display();

Animal\_ptr = &F; Animal\_ptr->Display();

Animal\_ptr = **new Cat**; Animal\_ptr->Display();

return 0;

}

Παρατηρήστε πως μπορούμε και δυναμικά να δημιουργήσουμε αντικείμενα με την εντολή new. Μπορείτε να επεκτείνετε τον προηγούμενο κώδικα ορίζοντας έναν πίνακα από pointers σε Animal και αναθέτοντας σε αυτόν διαδοχικά διάφορα αντικείμενα Cat, Dog και Fish.

### Άσκηση 1

Να επεκτείνετε την παραπάνω άσκηση ορίζοντας μέσα στην main έναν **πίνακα** από pointers σε Animal (μεγέθους 10).

Στην συνέχεια, στην main να υλοποιήσετε με loop ένα μενού μέσω του οποίου ο χρήστης θα επιλέγει τι είδος ζώου θα τοποθετήσει στον πίνακα κάθε φορά. Αν πχ. ο χρήστης επιλέξει σκύλο, τότε θα πρέπει να δημιουργήσετε ένα νέο αντικείμενο σκύλου (new Dog ) το οποίο θα πρέπει να τοποθετήσετε στην κατάλληλη θέση του πίνακα.

Όταν ο χρήστης τελειώσει με τις επιλογές του, με ένα άλλο loop να εκτυπώσετε στην οθόνη πληροφορίες για κάθε ένα ζώο που υπάρχει μέσα στον πίνακα.

## Καθαρά Δυναμικές – Εικονικές (Virtual) Συναρτήσεις – Abstract κλάσεις

Όπως διαπιστώσαμε στο προηγούμενο παράδειγμα όταν μια δυναμική συνάρτηση δεν ορίζεται για μια παραγόμενη κλάση, τότε στη θέση της θα εκτελεστεί η έκδοση που ορίζεται στη βασική κλάση. Σε πολλές όμως περιπτώσεις η συνάρτηση αυτή δεν εκτελεί καμιά εργασία ή απλώς μπορεί να εμφανίζει ένα μήνυμα. Επίσης υπάρχουν και οι περιπτώσεις που υπάρχει η βεβαιότητα ότι η συνάρτηση που θα εκτελέσει μια παραγόμενη κλάση είναι αυτή που ορίζεται από την ίδια και όχι από τη βασική της. Η λύση σε αυτές τις περιπτώσεις είναι η καθαρά δυναμική συνάρτηση.

Η γενική μορφή της διακήρυξης μιας καθαρά δυναμικής συνάρτησης είναι:

virtual τύπος επιστρ. τιμής όνομα συνάρτησης (κατάλογος παραμέτρων) = 0;

Η καθαρά δυναμική συνάρτηση δεν έχει ορισμό σχετιζόμενο με αυτή. Επομένως κάθε παραγόμενη κλάση πρέπει να έχει τη δική της έκδοση για τη συνάρτηση, αφού δεν μπορεί να χρησιμοποιήσει της βασικής, αλλιώς ο μεταγλωττιστής θα δώσει μήνυμα λάθους.

Η κλάση που περιέχει τουλάχιστον μια καθαρά δυναμική συνάρτηση ονομάζεται abstract και έχει το ιδιαίτερο χαρακτηριστικό ότι δεν μπορούμε να διακηρύξουμε αντικείμενά της.

Παράδειγμα3

#include <iostream> using namespace std;

class Shape{ public:

virtual float Area()=0; virtual float Perimeter()=0;

};

class Square:public Shape{ public:

Square(float s=0){side=s;}

float Area(){return side\*side;}

float Perimeter(){return 4\*side;} private:

float side;

};

class Rectangle:public Shape{ private:

float length,width; public:

Rectangle(float l=0,float w=0){length=l;width=w;} float Area(){return length\*width;}

float Perimeter(){return 2\*length+2\*width;}

};

int main()

{

Shape \*s[3];

s[0] = new Square(5.31);

s[1] = new Rectangle(3, 7.9);

cout<<"Area of square is " <<s[0]->Area() <<" square units.\n";

cout <<"Perimeter of rectangle is " <<s[1]->Perimeter()<<"units.\n"; system("pause");

return 0;

}

### Άσκηση 2

Δοκιμάστε να δημιουργήσετε ένα αντικείμενο της κλάσης Shape. Επιτρέπεται και γιατί;

### Άσκηση 3

Συνεχίζοντας το παραπάνω πρόγραμμα, να υλοποιήσετε και μια κλάση Circle που θα είναι και αυτή υποκλάση

της Shape.

Στην συνέχεια, να εισάγετε και ένα αντικείμενο τύπου Circle στον πίνακα s.

Τέλος, στην main, να υλοποιήσετε ένα loop το οποίο θα υπολογίζει το άθροισμα των περιμέτρων όλων των Shapes, καθώς και το μεγαλύτερο εμβαδό από όλα τα Shapes που βρίσκονται στον πίνακα s.

## Namespaces

Σε πολλές περιπτώσεις που πρέπει να συνδυαστεί κώδικας από διαφορετικούς προγραμματιστές, υπάρχει το ενδεχόμενο να χρησιμοποιούνται τα ίδια ονόματα για συναρτήσεις ή global μεταβλητές. Για να αποφευχθεί η πιθανότητα σύγκρουσης των ονομάτων, χρησιμοποιούμε τους χώρους ονομάτων, ή αλλιώς namespaces. Με αυτό τον τρόπο, ο κώδικας από διαφορετικούς προγραμματιστές μπαίνει σε ξεχωριστό namespace και απομονώνεται εκεί.

Στο παρακάτω παράδειγμα ορίζουμε δύο namespaces που περιέχουν την ίδια κλάση SUV. Για να χρησιμοποιήσουμε κάποια από τις δύο κλάσεις πρέπει να κάνουμε χρήση του scope operator ::

namespace ford

{

class SUV

{

...

};

}

namespace dodge

{

class SUV

{

...

};

}

int main()

{

ford::SUV s1;

dodge::SUV s2;

}

Τώρα μπορούμε να καταλάβουμε την γνωστή εντολή: using namespace std: σημαίνει ότι όλη η standard library της C++ έχει μπει στο namespace std. Συνεπώς, για να χρησιμοποιήσουμε κάτι από αυτήν την βιβλιοθήκη, π.χ. το αντικείμενο cout θα πρέπει να πούμε σε ποιό namespace βρίσκεται. Άρα θα έπρεπε να γράφουμε κάθε φορά:

std::cout << “aaaa”;

Αντί αυτού, γράφουμε μία φορά στην αρχή του προγράμματός μας:

using namespace std

....

cout << “aaaa”;

Συνέχεια από το παραπάνω παράδειγμα:

namespace ford

{

class SUV

{

...

};

class Compact

{

...

};

}

int main()

{

using namespace ford; // exposes SUV and Compact SUV s1 = new SUV();

...

}

## Static data και functions

Όταν ορίσουμε μια μεταβλητή της κλάσης μας ως στατική (static), τότε αυτή η μεταβλητή είναι κοινή για όλα τα αντικείμενα αυτής της κλάσης. Δηλαδή υπάρχει μόνο ένα αντίγραφο αυτής της μεταβλητής συνολικά, και αν αλλάξει η τιμή της μεταβλητής αυτής τότε η αλλαγή αυτή είναι ορατή για όλα τα αντικείμενα της ίδιας κλάσης.

Οι static μεταβλητές μπορούν να χρησιμοποιηθούν για αποθήκευση κοινής πληροφορίας μεταξύ όλων των αντικειμένων μιας κλάσης, όπως φαίνεται και στο παρακάτω παράδειγμα, όπου η μεταβλητή total μετρά τα συνολικά αντικείμενα τύπου Counter που υπάρχουν κάθε στιγμή. Μια στατική μεταβλητή υπάρχει πριν ακόμα δημιουργηθεί ένα αντικείμενο της κλάσης, και αρχικοποιείται έξω από αυτήν.

Μπορούμε επίσης και να δηλώσουμε και συναρτήσεις μιας κλάσης ως static. Τότε όμως αυτές οι συναρτήσεις μπορούν να προσπελάσουν μόνο static δεδομένα.

#include <iostream>

using namespace std;

class Counter

{

private:

static int total; //total objects of this class

// (declaration only)

int id; //ID number of this object

public:

Counter() //no-argument constructor

{

total++; //add another object

id = total; //id equals current total

}

~Counter() //destructor

{

total--;

cout << "Destroying ID number " << id << endl;

}

static void showtotal() //static function

{

cout << "Total is " << total << endl;

}

void showid() //non-static function

{

cout << "ID number is " << id << endl;

}

};

int Counter::total = 0; //important: definition of total

int main()

{

Counter g1; Counter::showtotal();

Counter g2, g3; Counter::showtotal();

g1.showid();

g2.showid();

g3.showid();

system("pause");

return 0;

}

## Παράρτημα - Παράδειγμα με window class hierarchy

Το πιο κλασσικό παράδειγμα χρήσης των virtual functions είναι κατά τον παραθυρικό προγραμματισμό. Οι διάφορες βιβλιοθήκες που μπορούμε να χρησιμοποιήσουμε για να δημιουργήσουμε παράθυρα στις εφαρμογές μας, έχουν ορίσει μια ιεραρχία κλάσεων που παριστάνουν τα διαφορετικά components (windows, panels, buttons, combo boxes κλπ.). Γενικώς όλες αυτές οι κλάσεις κληρονομούν από μια αρχική κλάση, ψηλά στην ιεραρχία (ας την πούμε Component).

Στο παρακάτω απλό παράδειγμα, η κλάση Component ορίζει μέσα της μόνο μια συνάρτηση, την **pure virtual**

function paint(). Κάθε κλάση που κληρονομεί από αυτήν, θα πρέπει να την υλοποιήσει. Έτσι, οι κλάσεις CommandButton και TextBox υλοποιούν αυτή την συνάρτηση. Όταν κληθεί αυτή η συνάρτηση για κάποιο από αυτά τα αντικείμενα, θα πρέπει αυτό να ζωγραφίσει τον εαυτό του στην οθόνη.

Στην συνάρτηση main, δημιουργούμε ένα παράθυρο, το οποίο περιέχει δύο Components: ένα TextBox και ένα CommandButton. Παρατηρούμε ότι έχουμε ορίσει ένα πίνακα που περιέχει pointers σε Component (την μητρική κλάση), και ότι σε αυτό τον πίνακα αποθηκεύουμε τους δείκτες των δύο components. Έχουμε με αυτό τον τρόπο την δυνατότητα κοινού χειρισμού όλων των components του παραθύρου.

Αν μετακινήσουμε το παράθυρο αυτό, θα πρέπει να το ζωγραφίσουμε πάλι στην οθόνη. Αυτό γίνεται δίνοντας την εντολή paint σε κάθε ένα από τα components που βρίσκονται πάνω στο παράθυρο. Για να γίνει αυτό, αρκεί να κάνουμε ένα loop σε όλο τον πίνακα και να καλέσουμε την paint() για όλα τα στοιχεία του πίνακα.

#include "stdafx.h"

#include <iostream>

using namespace std;

class Component

{

public:

virtual void paint() = 0;

};

class CommandButton : public Component

{

public:

void paint()

{

endl;

}

};

cout<<"Derived class Command Button - Overridden C++ virtual function" <<

class TextBox : public Component

{

public:

void paint()

{

cout<<"Derived class TextBox - Overridden C++ virtual function" << endl;

}

};

int main()

{

Component \*components[2];

components[0] = new TextBox(); components[1] = new CommandButton();

for(int i=0;i<2;i++)

components[i]->paint();

system("pause");

return 0;

}

|  |
| --- |
| **Ανοικτά Ακαδημαϊκά Μαθήματα****Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας** |
| **Τέλος Ενότητας** |
| **Χρηματοδότηση*** Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Αθήνας**» 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.

 |

**Σημειώματα**

**Σημείωμα Αναφοράς**

Copyright ΤΕΙ Αθήνας, Κλειώ Σγουροπούλου, 2014. Κλειώ Σγουροπούλου. «Αντικειμενοστρεφής Προγραμματισμός». Ενότητα 6: «Συναρτήσεις». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: [ocp.teiath.gr](https://ocp.teiath.gr/).

**Σημείωμα Αδειοδότησης**

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

* που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
* που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
* που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

**Διατήρηση Σημειωμάτων**

* Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
* Το Σημείωμα Αναφοράς
* Το Σημείωμα Αδειοδότησης
* Τη δήλωση Διατήρησης Σημειωμάτων
* Το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους.