ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

# ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

## Ενότητα 6: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ


ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

# 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.

# Χρηματοδότηση


- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» 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.

# Ενότητα 6

## ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

**ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ**

# Περιεχόμενα ενότητας

1. Δείκτες
2. Δείκτες – δυναμική διαχείριση μνήμης
3. Δείκτες για αντικείμενα
4. Πίνακες δεικτών προς αντικείμενα
5. Ο δείκτης this
6. Πολυμορφισμός (Polymorphism)
7. Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης
8. Δείκτης της παράγωγης κλάσης σε αντικείμενο της βασικής κλάσης
9. Κλήση συνάρτησης παράγωγης κλάσης μέσω δείκτη βασικής κλάσης
10. Εικονικές συναρτήσεις (virtual functions)
11. Εικονικές και μη εικονικές συναρτήσεις
12. Γνήσιες εικονικές συναρτήσεις
13. Αφηρημένες κλάσεις
14. Μελέτη περίπτωσης εικονικών συναρτήσεων
15. Εικονικές συναρτήσεις αποδόμησης
16. Μελέτη περίπτωσης πολυμορφισμού

# Σκοποί ενότητας

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Δείκτες

***Ο δείκτης είναι μία μεταβλητή που περιέχει τη διεύθυνση μίας άλλης μεταβλητής.***

**Η χρήση δεικτών είναι αρκετά συνηθισμένη σε αρκετές γλώσσες προγραμματισμού όπως και στην C++. Αν και πολλές λειτουργίες μπορούν να υλοποιηθούν και με άλλους τρόπους, εν τούτοις σε αρκετές περιπτώσεις οι δείκτες γίνονται ένα προγραμματιστικό εργαλείο για αύξηση της δύναμης της γλώσσας προγραμματισμού, όπως είναι κάποιες δομές δεδομένων (οι συνδεδεμένες λίστες, τα δυαδικά δένδρα κ.λ.π.).**

# Δείκτες

**Μερικές από τις πλέον συνηθισμένες χρήσεις τους είναι:**

- **Προσπέλαση στοιχείων πίνακα**
- **Μεταβίβαση ορισμάτων σε συνάρτηση, όταν η συνάρτηση πρέπει να αλλάξει το αρχικό όρισμα**
- **Μεταβίβαση πινάκων και αλφαριθμητικών σε συναρτήσεις**
- **Απόκτηση μνήμης από το σύστημα**
- **Δημιουργία δομών δεδομένων όπως οι συνδεδεμένες λίστες**

**Εάν κατά τη δήλωση ενός δείκτη επιθυμούμε να μην έχει απροσδιόριστη τιμή, τότε τον αρχικοποιούμε στο NULL και όχι στο 0:**

```
int *my_pointer = NULL;
```

# Δείκτες - δυναμική διαχείριση μνήμης

Οι δείκτες λειτουργούν με βάση την έννοια της εμμεσότητας. Ένας δείκτης είναι μία μεταβλητή, στην οποία έχει αποθηκευτεί ως τιμή μία διεύθυνση της μνήμης του υπολογιστή. Στη διεύθυνση αυτή αντιστοιχεί ένα ορισμένο περιεχόμενο της μνήμης, που είναι η τιμή ενός ορισμένου δεδομένου. Τότε, ο δείκτης αυτός έχει δυνατότητα έμμεσης πρόσβασης στην τιμή αυτού του δεδομένου.

Όταν χρειαζόμαστε μνήμη για την αποθήκευση πολλών στοιχείων, συνήθως χρησιμοποιούμε πίνακες. Οι πίνακες όμως έχουν το μειονέκτημα ότι πρέπει να καθορισθεί το μέγεθός τους και μάλιστα πριν την εκτέλεση του προγράμματος. Σε πολλές, όμως, περιπτώσεις δε γνωρίζουμε πόση μνήμη θα χρειαστούμε εκ των προτέρων.

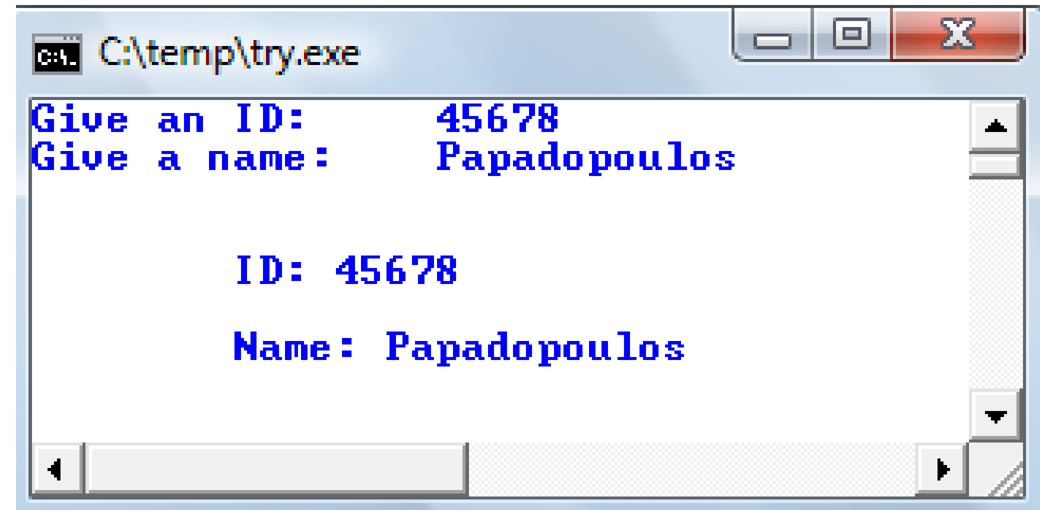
Η C++ μας παρέχει, μέσω νέων τελεστών, τη δυνατότητα να δεσμεύσουμε και να αποδεσμεύσουμε τμήματα μνήμης, πέραν των μεθόδων που έχουμε μάθει. Οι τελεστές αυτοί είναι οι **new** και **delete**.

# Δείκτες

```
#include <cstdlib>
#include <iostream>

using namespace std;

struct  person                {
            int id;    char name[20];
};
main()    {
            struct person *ptr;
            ptr = new struct person;
            cout << "Give an ID:\t";    cin >> ptr->id;
            cout << "Give a name:\t"; cin >> ptr->name;
            cout << "\n\n\tID: " << ptr->id;
            cout << "\n\n\tName: " << ptr->name;
            delete ptr;
}
```




*theory_6_point_1.cpp*

# Δείκτες

Η χρήση του τελεστή new είναι παρόμοια με τη χρήση της συνάρτησης malloc() της C. Η νέα προσέγγιση με τον new είναι καλύτερη γιατί επιστρέφει έναν δείκτη για τον κατάλληλο τύπο δεδομένων, ενώ ο δείκτης της malloc() πρέπει να προσαρμοσθεί (*casting*) στον κατάλληλο τύπο.

Παρατήρηση: Σε παλαιότερους μεταγλωττιστές της C++, εάν δεν υπήρχε επαρκής διαθέσιμη μνήμη για τη δημιουργία μίας νέας μεταβλητής ή αντικειμένου, στον δείκτη αποδίδετο το NULL. Στην πρότυπη C++ εάν δεν υπάρχει επαρκής διαθέσιμη μνήμη, ο τελεστής new τερματίζει το πρόγραμμα.

# Δείκτες για αντικείμενα

Οι δείκτες, πέρα από τους συνήθεις τύπους δεδομένων, μπορούν να δείχνουν και σε αντικείμενα. Αν κατά τη στιγμή που γράφουμε ένα πρόγραμμα, δε γνωρίζουμε πόσα αντικείμενα θα δημιουργήσουμε, τότε μπορούμε να χρησιμοποιήσουμε τον τελεστή new. Ο new επιστρέφει έναν δείκτη σε ένα ανώνυμο αντικείμενο και μπορούμε έτσι να δημιουργήσουμε αντικείμενα κατά τη διάρκεια εκτέλεσης του προγράμματος.

```
class Person {
private:
  int id;
  char name[20];
public:
  void readData()  {
    cout << "Δώσε κωδικό:";
    cin >> id;
    cout << "Δώσε ένα όνομα:";
    cin >> name;  }
  void printData()  {
    cout  <<  "Κωδικός:  "  <<  id
<< endl;
cout  <<  "Όνομα:  "  <<  name
<< endl;  }
};
```

# Δείκτες για αντικείμενα

```
main()  {
    Person  *p;
    p = new Person;         p→readData();    p→printData();
}
```

Δηλώνεται ένας δείκτης p σε ένα αντικείμενο τύπου Person, με τον τελεστή new δεσμεύεται μνήμη για το αντικείμενο και ορίζεται ο δείκτης p να δείχνει στο συγκεκριμένο αντικείμενο.

Για να αναφερθούμε στις συναρτήσεις-μέλη του αντικειμένου χρησιμοποιούμε τον τελεστή προσπέλασης μέλους -> (τελεστής βέλους).

Κατά τη δέσμευση μνήμης ενός αντικειμένου μπορεί να χρησιμοποιηθεί δομητής με ορίσματα, όπως ακριβώς γίνεται και στη συνήθη δήλωση αντικειμένου χωρίς δυναμική δέσμευση μνήμης. Π.χ. για δομητή MyClass(int x, int y) μίας κλάσης MyClass:

```
MyClass *myPointer;
myPointer = new MyClass(4, -17);
```

# Πίνακες δεικτών προς αντικείμενα

*Όταν σε ένα πρόγραμμα δημιουργούμε πολλά αντικείμενα, είναι πιο ευέλικτο αντί να τοποθετήσουμε τα ίδια τα αντικείμενα στον πίνακα, να χρησιμοποιήσούμε έναν πίνακα δεικτών προς αυτά:*

```
class Person      {
private:
        int id;     char name[20];
public:
        void readData()  {
                cout << "Δώσε κωδικό:";          cin >> id;
                cout << "Δώσε ένα όνομα:";       cin >> name;
        }
        void printData()  {
                cout << "Κωδικός: " << id << endl;
                cout << "Όνομα: " << name << endl;
        }
};
```

# Πίνακες δεικτών προς αντικείμενα

```
main()
{
    Person  *perspin[100];
    int i = 0, j;
    char choice;
    do
    {
    perspin[i] = new Person;
            perspin[i] →readData();
            i++;
            cout << “More?”;
            cin >> choice;
    }       while (choice == ‘y’);
    for (j=0; j<i; j++)   perspin[j] →printData();
}
```

TEI ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Ο δείκτης this

Οι συναρτήσεις-μέλη κάθε αντικειμένου έχουν πρόσβαση σε έναν «αόρατο» δείκτη που ονομάζεται this. Έτσι κάθε συνάρτηση-μέλος μπορεί να ανακαλύψει τη διεύθυνση του αντικειμένου στο οποίο ανήκει. Επιπλέον ο δείκτης this μπορεί να χρησιμοποιηθεί όπως κάθε άλλος δείκτης για κάποιο αντικείμενο και έτσι μπορεί να προσπελάσει δεδομένα στο αντικείμενο που αναφέρεται. Βέβαια, για λόγους ευκολίας συνήθως η αναφορά στα δεδομένα αυτά γίνεται άμεσα, παραλείποντας τον δείκτη this. Μία πιο πρακτική χρήση του this προκύπτει όταν μία συνάρτηση-μέλος ενός αντικειμένου θέλει να επιστρέψει το αντικείμενο που την έχει καλέσει.

Γενικά μπορούμε να ισχυριστούμε ότι, για λόγους κατανόησης, είναι σωστό να γίνεται η χρήση του δείκτη this. Για λόγους όμως οικονομίας είναι προτιμότερο να παραλείπεται και η αναφορά στα δεδομένα του αντικειμένου να γίνεται άμεσα.

# Ο δείκτης this

Το παρακάτω παράδειγμα δείχνει πώς μπορεί να γίνει η χρήση του **this**:

```
#include <iostream>
class Employee   {
private:
        float mikta;        int categ;
public:
        Employee()        {
                this→mikta = 0;
                this→categ = 0;
        }
        Employee(float mikta0, int categ0)
       {
                this→mikta = mikta0;
                this→categ = categ0;
       }
```

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Ο δείκτης this

```
        void showAddress()          {
                cout << "The address of the object is " << this << endl;
        }
        float calcTax()      {
                float foros;
                if (this→categ == 1)     foros = this→mikta * 0.15;
                else        foros = this→mikta * 0.3;
                return foros;
        }
        float calcSalary(){
                float misthos;
                misthos = this→mikta - this→calcTax();
                return misthos;
        }
};
```

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Ο δείκτης this

```
main()
{
   Employee emp(1000,1);

   emp.showAddress();
   cout << "The salary is: " << emp.calcSalary() << endl;
}
```

# Πολυμορφισμός (Polymorphism)

- Αναφέρεται στη δυνατότητα να συνδυάζουμε πολλά περιεχόμενα σε ένα όνομα συνάρτησης μέσω του μηχανισμού της δυναμικής σύνδεσης. Επομένως **πολυμορφισμός, δυναμική σύνδεση και εικονικές συναρτήσεις** είναι ουσιαστικά εκφάνσεις της ίδιας έννοιας, όπως θα αναλύσουμε ακολούθως.
- Απαιτείται η ύπαρξη μίας ιεραρχίας κλάσεων.
- Χρησιμοποιούμε αντικείμενα που ανήκουν στην ίδια ιεραρχία κλάσεων ως να ήταν όλα αντικείμενα της βασικής κλάσης.
- Επιτρέπει την αποτελεσματική επέκταση των προγραμμάτων, καθώς νέες κλάσεις μπορούν εύκολα να προστεθούν και να υποστούν επεξεργασία όπως και οι υπάρχουσες.

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

Υπάρχουν τρεις διαφορετικοί τρόποι χειρισμού των δεικτών σε αντικείμενα μίας ιεραρχίας κλάσεων:

- Δείκτης βασικής κλάσης σε αντικείμενο της βασικής κλάσης
- Δείκτης παράγωγης κλάσης σε αντικείμενο της παράγωγης κλάσης
- Δείκτης βασικής κλάσης σε αντικείμενο της παράγωγης κλάσης

Η κλήση συναρτήσεων εξαρτάται από την κλάση του δείκτη και δεν εξαρτάται από το αντικείμενο στο οποίο δείχνει ο δείκτης. Αυτό μπορεί να αλλάξει με τη χρήση των εικονικών συναρτήσεων.

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
#include <cstdlib>
#include <iostream>
#include <iomanip>

using namespace std;

const double my_pi=3.14159;

class Point
{
private:
        int x;  // Τετμημένη
        int y;  // Τεταγμένη
```

```
public:
        Point(int x,int y);
        void setX(int x);
        int getX();
        void setY(int y);
        int getY();
        void print(); // εκτύπωση
                      // σημείου
}; // Τέλος της κλάσης Point
```

theory_6_polym_1.cpp

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
Point :: Point(int xValue,int yValue) : x(xValue), y(yValue)
{}
void Point :: setX(int xValue)      {
  x = xValue;
}
int Point :: getX()           {
  return x;
}
void Point :: setY(int yValue)      {
  y = yValue;
}
int Point :: getY()           {
  return y;
}
void Point :: print()         {
  cout << '[' << x << ", " << y << ']';
}
```

*Δομητής της Point*

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
class Circle : public Point
{
private:
        double radius;  // Ακτίνα του κύκλου
public:
        Circle(int = 0,int = 0, double radius = 0.0);
        void setRadius(double radius);
        double getRadius();
        double getDiameter(); // Υπολογίζει τη διάμετρο
        double getCircumference();   // Υπολογίζει την περιφέρεια
        double getArea();            // Υπολογίζει το εμβαδόν
        void print();
}; // Τέλος της κλάσης Circle
```

*Κληρονομεί την Point.*

**Επαναπροσδιορισμός της συνάρτησης print**

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
Circle::Circle(int xValue,int yValue,double radiusValue) :
                                        Point(xValue,yValue)
{
  setRadius(radiusValue); // ή αλλιώς ο κώδικας της setRadius
}
void Circle::setRadius(double radiusValue)
{
  if (radiusValue < 0.0) radius = 0.0;
  else radius = radiusValue;
}
double Circle::getRadius()
{
  return radius;
}
```

***Δομητής της Circle, όπου αρχικοποιούνται και οι κληρονο-μούμενες μεταβλητές από την Point.***

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
double Circle::getDiameter()        {
        return 2 * getRadius();
}
double Circle::getCircumference()           {
        return my_pi * getDiameter();
}
double Circle::getArea()  {
        return my_pi * getRadius() * getRadius();
}
void Circle::print()        {
        cout << "Center = ";
        Point::print();
        cout << "; Radius = " << getRad
}
```

*Εκ νέου ορισμός της συνάρτησης* ***print****, ώστε να εκτυπώνεται* ***και*** *η ακτίνα του κύκλου.*

*Καλείται η συνάρτηση* ***print*** *της βασικής ως προς την* ***Cycle*** *κλάσης* ***Point****, με χρήση του τελεστή* ***::****.*

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
main()
{
    Point point(30,50), *pointPtr; // Δείκτης βασικής κλάσης
    Circle circle(120,89,2.7), *circlePtr; // Δείκτης παράγωγης
                                           // κλάσης
    cout << fixed << setprecision( 2 );
    cout << "Print point and circle objects:" << "\nPoint: ";
    point.print();
    cout << "\nCircle: ";
    circle.print();
    pointPtr = &point;
    cout << "\n\nCalling print with base-class pointer to "
     << "\nbase-class object invokes base-class print "
     << "function:\n";
    pointPtr → print();
```

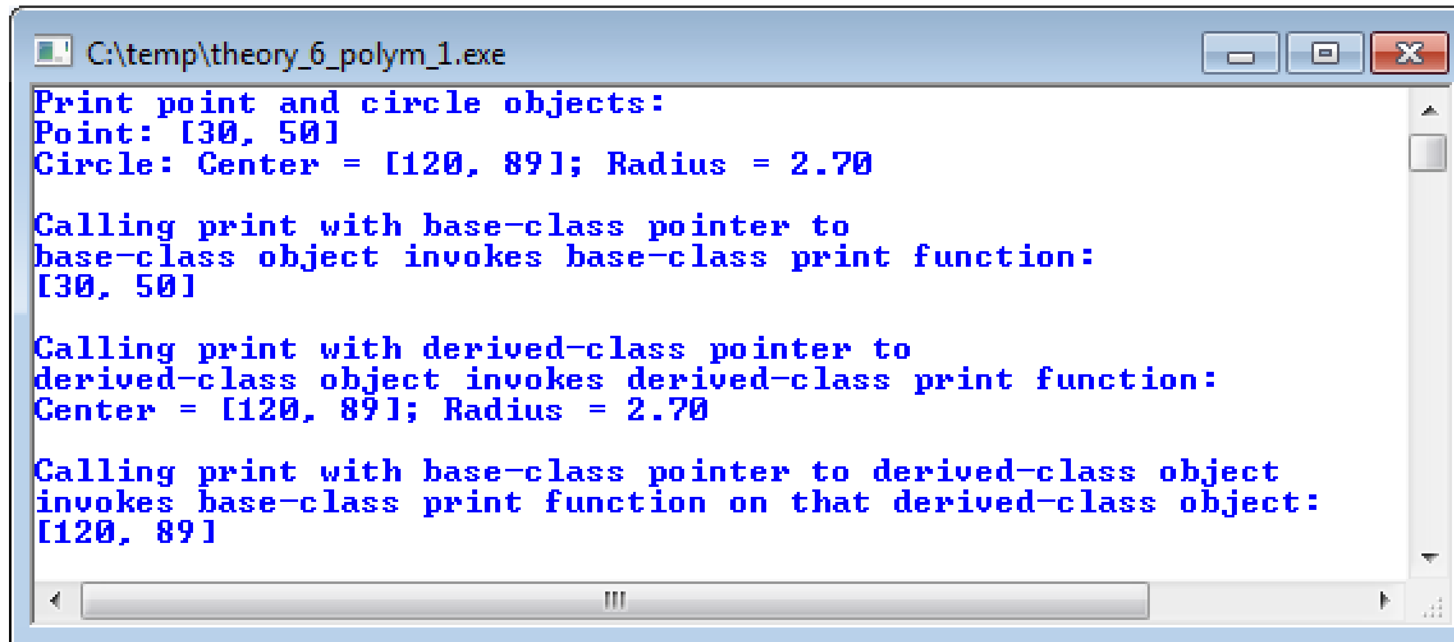
*Η κλήση της συνάρτησης* ***print*** *γίνεται με χρήση αντικειμένων, τα οποία ανήκουν στην ίδια κλάση, έτσι ώστε να καλείται η κατάλληλη* ***print****.*

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης

```
   circlePtr = &circle;
   cout << "\n\nCalling print with derived-class pointer to "
        << "\nderived-class object invokes derived-class "
        << "print function:\n";
   circlePtr → print();
   pointPtr = &circle;
   cout << "\n\nCalling print with base-class pointer to "
        << "derived-class object\ninvokes base-class print "
        << "function on that derived-class object:\n";
   pointPtr → print();
}
```

*Επιτρέπεται ένα δείκτης βασικής κλάσης να «δείχνει» σε ένα αντικείμενο παράγωγης κλάσης (**Circle** "is a" **Point**). Ωστόσο, καλεί τη συνάρτηση **print** της κλάσης **Point**, όπως καθορίζεται από τον τύπο του δείκτη. Οι εικονικές συναρτήσεις επιτρέπουν να αλλαχθεί αυτό.*

# Κλήση συναρτήσεων της βασικής κλάσης από αντικείμενα της παραγόμενης κλάσης




Ένας δείκτης της βασικής κλάσης μπορεί να χρησιμοποιηθεί μόνο για πρόσβαση στα μέλη ενός αντικειμένου της παράγωγης κλάσης που έχουν κληρονομηθεί από τη βασική κλάση.

# Δείκτης της παράγωγης κλάσης σε αντικείμενο της Βασικής κλάσης

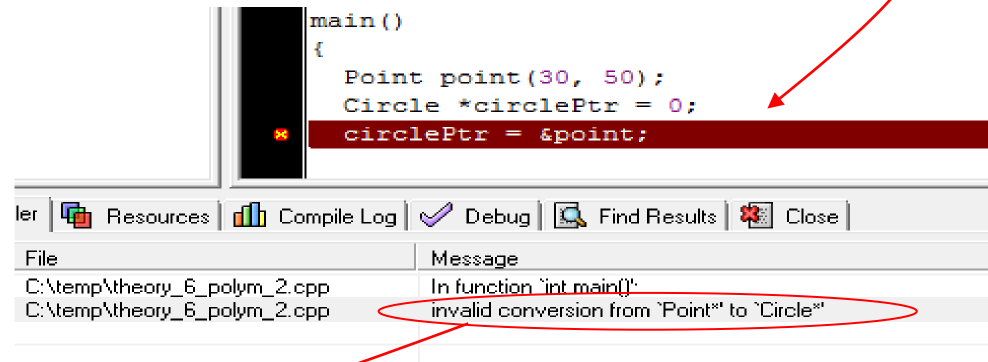
**Στο προηγούμενο παράδειγμα είδαμε δείκτη της βασικής κλάσης σε αντικείμενο της παράγωγης κλάσης (Circle "is a" Point).**

**Εάν χρησιμοποιηθεί δείκτης παράγωγης κλάσης σε αντικείμενο της βασικής κλάσης θα προκύψει σφάλμα μεταγλώττισης (Point is not a Circle). Η κλάση Circle διαθέτει ιδιότητες/λειτουργίες που δεν διαθέτει η κλάση Point (π.χ. setRadius).**

**Μπορούμε να κάνουμε μετατροπή τύπου (casting) στη διεύθυνση του αντικειμένου της βασικής κλάσης σε δείκτη παράγωγης κλάσης: Ονομάζεται downcasting και επιτρέπει την προσπέλαση λειτουργικότητας της παράγωγης κλάσης** (λεπτομέρειες σε επόμενες διαφάνειες)**.**

# Δείκτης της παράγωγης κλάσης σε αντικείμενο της Βασικής κλάσης

```
main()
{
        Point point(30, 50);
        Circle *circlePtr;
        circlePtr = &point;
}
```




theory_6_polym_2.cpp

# Κλήση συνάρτησης παράγωγης κλάσης μέσω δείκτη βασικής κλάσης

Ο δείκτης βασικής κλάσης μπορεί μεν να δείχνει σε αντικείμενο της παράγωγης κλάσης αλλά μπορεί να καλέσει μόνο τις μεθόδους της βασικής κλάσης. Η κλήση μεθόδων της παράγωγης κλάσης συνιστά λάθος, καθώς οι μέθοδοι αυτές δεν ορίζονται στη βασική κλάση. Ο τύπος δεδομένων ενός δείκτη προσδιορίζει τι μεθόδους μπορούμε να καλέσουμε.

```
main()  {
        Point *pointPtr;
        Circle circle( 120, 89, 2.7 );
        pointPtr = &circle;
        int x = pointPtr → getX();
        int y = pointPtr → getY();
        pointPtr → setX( 10 );
        pointPtr → setY( 10 );
        pointPtr → print();
```

*theory_6_polym_3.cpp*

# Κλήση συνάρτησης παράγωγης κλάσης μέσω δείκτη βασικής κλάσης

```
double radius = pointPtr->getRadius();
pointPtr → setRadius( 33.33 );
double diameter = pointPtr → getDiameter();
double circumference = pointPtr → getCircumference();
double area = pointPtr → getArea();
}
```

*Αυτές οι συναρτήσεις ορίζονται μόνο στην κλάση* ***Circle****. Ωστόσο ο* ***pointPtr*** *είναι κλάσης* ***Point****.*

```
int y = pointPtr -> getY();
pointPtr -> setX( 10 );
pointPtr -> setY( 10 );
pointPtr -> print();
double radius = pointPtr->getRadius();
pointPtr -> setRadius( 33.33 );
```

mpiler | Resources | Compile Log | Debug | Find Results | Close

| File | Message |
|---|---|
| C:\temp\theory_6_polym_3.cpp | In function `int main()': |
| C:\temp\theory_6_polym_3.cpp | 'class Point' has no member named 'getRadius' |
| C:\temp\theory_6_polym_3.cpp | 'class Point' has no member named 'setRadius' |
| C:\temp\theory_6_polym_3.cpp | 'class Point' has no member named 'getDiameter' |
| C:\temp\theory_6_polym_3.cpp | 'class Point' has no member named 'getCircumference' |
| C:\temp\theory_6_polym_3.cpp | 'class Point' has no member named 'getArea' |

*theory_6_polym_3.cpp*

# Εικονικές συναρτήσεις (virtual functions)

Ο κανόνας είναι ο τύπος του δείκτη να καθορίζει τι μέθοδοι μπορούν να κληθούν. Οι εικονικές συναρτήσεις μπορούν να τον αλλάξουν, ώστε το αντικείμενο (και όχι ο δείκτης) να καθορίζει τι μέθοδοι μπορούν να κληθούν, έτσι ώστε μέσω ενός δείκτη της βασικής κλάσης να καλούνται συναρτήσεις-μέλη των παράγωγων κλάσεων.

Εικονική είναι μία συνάρτηση που στην πραγματικότητα δεν υπάρχει, αλλά εμφανίζεται ως πραγματική σε ορισμένα τμήματα ενός προγράμματος. Μία τέτοια συνάρτηση θα ήταν χρήσιμη αν είχαμε ένα πλήθος αντικειμένων από διαφορετικές κλάσεις και θέλαμε να τα τοποθετήσουμε όλα σε μία λίστα, ώστε να εκτελέσουμε κάποια συγκεκριμένη λειτουργία χρησιμοποιώντας την ίδια κλήση συνάρτησης.

```
class A            {
public:
        virtual void display()        {
                cout << "Base class.";
        }
};
```

theory_6_vf_1.cpp

# Εικονικές συναρτήσεις

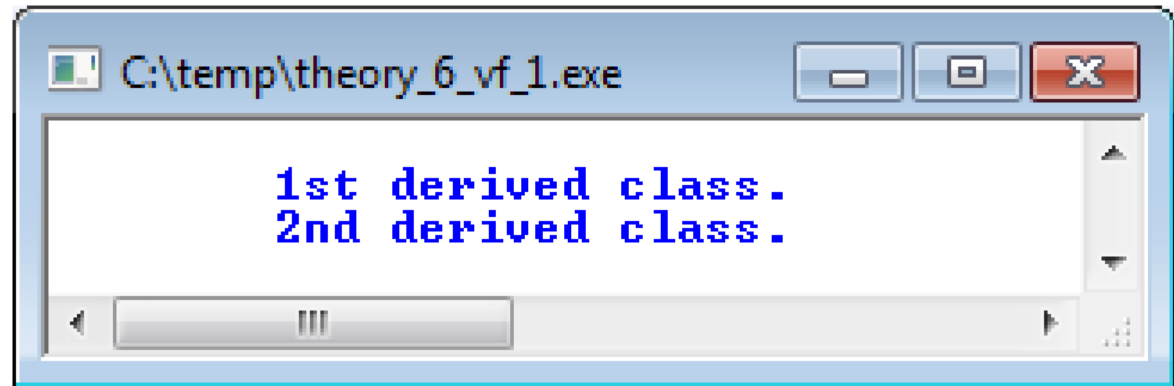
```
class Par1 : public A    {
public:
        void display()    {
         cout << "1st derived class.";
        }
};

class Par2 : public A    {
public:
        void display()    {
        cout << "2nd derived class.";
        }
};
```

Στις παράγωγες κλάσεις δε χρησιμοποιείται το προσδιοριστικό **virtual** για τις εικονικές συναρτήσεις.

# Εικονικές συναρτήσεις

```
main( )
{
   Par1  p1;
   Par2  p2;
   A *ptr;
   ptr = &p1;
   ptr→display();
   ptr = &p2;
   ptr→display();
}
```




Η βασική κλάση A και οι δύο παράγωγες της A, Par1 και Par2 έχουν μία συνάρτηση με το ίδιο όνομα, display(). Οι συναρτήσεις προσπελαύνονται με χρήση δεικτών. Θα πρέπει να σημειωθεί ότι στη βασική κλάση μπροστά από το δηλωτικό της συνάρτησης έχει προστεθεί η δεσμευμένη λέξη virtual.

# Εικονικές συναρτήσεις

Εξ αιτίας αυτής της προσθήκης, όταν εκτελείται η πρόταση

**ptr→display();**

καλείται η συνάρτηση της παράγωγης κλάσης, ενώ αν παραλειπόταν η λέξη **virtual** θα καλείτο η συνάρτηση **display()** της βασικής κλάσης. Δηλαδή **όταν μία συνάρτηση δηλωθεί virtual ο μεταγλωττιστής επιλέγει τη συνάρτηση με βάση το περιεχόμενο του δείκτη και όχι τον τύπο του δείκτη, ακόμη κι αν η εικονική συνάρτηση χρησιμοποιείται έμμεσα, καλούμενη στον ορισμό μίας κληρονομούμενης συνάρτησης**. Αυτή η μέθοδος καθορισμού μίας εικονικής συνάρτησης που θα χρησιμοποιηθεί είναι γνωστή ως **δυναμική ή καθυστερημένη σύνδεση (dynamic/late binding)**.

Μία κλάση που περιέχει εικονικές συναρτήσεις καλείται και **πολυμορφική κλάση**.

# Εικονικές συναρτήσεις

Όταν ο ορισμός μίας εικονικής συνάρτησης αλλάζει μέσα σε μία παράγωγη κλάση, τότε αναφέρεται ότι ο ορισμός συνάρτησης υπερβαίνεται (overridden). Στη βιβλιογραφία της C++ γίνεται διαχωρισμός μεταξύ των ορίων επαναπροσδιορισμός και υπέρβαση.

Και οι δύο όροι αναφέρονται στην αλλαγή του ορισμού μίας συνάρτησης μέσα σε μία παράγωγη κλάση. Αν η συνάρτηση είναι μία εικονική συνάρτηση, αυτή η ενέργεια καλείται υπέρβαση. Αν η συνάρτηση δεν είναι μία εικονική συνάρτηση, καλείται επαναπροσδιορισμός. Οι δύο αυτές περιπτώσεις αντιμετωπίζονται διαφορετικά από τον μεταγλωττιστή.

Η εικονικές συναρτήσεις κληρονομούνται, επομένως θεωρούνται εικονικές και σε επόμενο επίπεδο κληρονομικότητας.

# Εικονικές συναρτήσεις

**Κανόνες για τη χρήση εικονικών συναρτήσεων:**

**(α) Μία εικονική συνάρτηση πρέπει να είναι μέλος μίας κλάσης και να μην είναι στατική.**

**(β) Κάθε «έκδοση» της εικονικής συνάρτησης στις παράγωγες κλάσεις πρέπει να έχει το ίδιο «αποτύπωμα» με εκείνο της βασικής κλάσης, δηλαδή πρέπει να είναι του ίδιου τύπου και να έχει τον ίδιο αριθμό και τύπο παραμέτρων.**

**(γ) Οι εικονικές συναρτήσεις κληρονομούνται: Υποθέτουμε ότι η κλάση BC διαθέτει μία εικονική συνάρτηση vf() και η DC παράγεται από την κλάση BC. Για μία κλάση DC_to_DC, η οποία παράγεται από την κλάση DC, η συνάρτηση vf() θεωρείται εικονική παρά το γεγονός ότι μέσα στην κλάση DC δεν έχει δηλωθεί ως εικονική μέσω του προσδιοριστικού virtual.**

**(δ) Δεν επιτρέπεται σε μία συνάρτηση δόμησης να είναι εικονική, ενώ οι συναρτήσεις αποδόμησης μπορούν να δηλωθούν ως εικονικές** *(θα εξεταστεί στη συνέχεια)*.

# Εικονικές συναρτήσεις

**Πότε χρησιμοποιείται μία εικονική συνάρτηση:**

**Είναι φανερό ότι οι εικονικές συναρτήσεις έχουν πλεονεκτήματα και γι' αυτόν τον λόγο στη Java όλες οι συνάρτήσεις είναι εικονικές. Ωστόσο υπάρχει ένα μεγάλο λειτουργικό κόστος μετατρέποντας μία συνάρτηση σε εικονική, επειδή έτσι χρησιμοποιείται περισσότερη μνήμη και το πρόγραμμα εκτελείται πιο αργά από όσο θα εκτελείτο εάν η συνάρτηση δεν ήταν εικονική.**

**Αυτή είναι η αιτία που οι σχεδιαστές της C++ έδωσαν στον προγραμματιστή τον έλεγχο να αποφασίζει ποιες συναρτήσεις μέλη είναι εικονικές και ποιες όχι. Στην περίπτωση που η ταχύτητα είναι σημαντικός παράγοντας, τότε η χρήση εικονικών συναρτήσεων πρέπει να γίνεται με φειδώ.**

***Πάντως θα πρέπει να σημειωθεί ότι η ιδιότητα των εικονικών συναρτήσεων επιτρέπει σε ένα πρόγραμμα να καλέσει συναρτήσεις που δεν υφίστανται καν την ώρα που ο κώδικας μεταγλωττίζεται.***

# Εικονικές συναρτήσεις

Παράδειγμα: Χρήση εικονικών συναρτήσεων για να προσπελασθεί με ενιαίο τρόπο (επαναληπτική πρόταση) πλήθος αντικειμένων που προέρχονται από διαφορετικές παράγωγες κλάσεις της ίδιας βασικής κλάσης.

```
#include <iostream>
using namespace std;
class polygon  {
protected:
        float width,height;
public:
        void set(float a, float b)  {
                width = a;              height = b;
        }
        virtual void show()  {
                cout << "Polygon" << endl;
        }
};
```

theory_6_vf_2.cpp

# Εικονικές συναρτήσεις

```
class rectangle : public polygon  {
public:
        rectangle(float p, float y)
        {
                set(p,y);
        }
        float area()
        {
                return(width * height);
        }
        void show()
        {
          cout << "Rectangle:\t" << width << " x " << height << endl;
        }
};
```

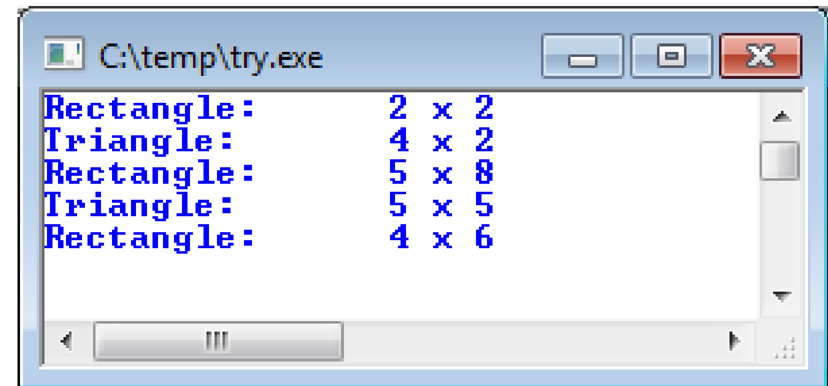
# Εικονικές συναρτήσεις

```
class triangle : public polygon  {
public:
        triangle(float p, float y)
        {
                set(p,y);
        }
        float area()
        {
                return(width * height / 2);
        }
        void show()
        {
           cout << "Triangle:\t" << width << " x " << height << endl;
        }
};
```

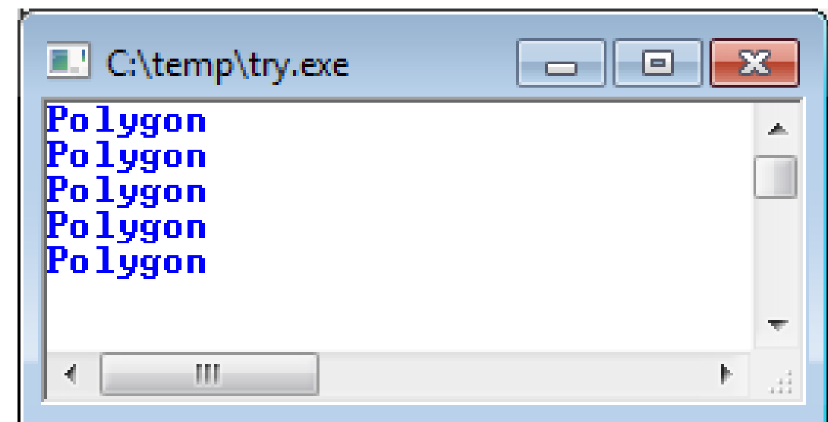
# Εικονικές συναρτήσεις

```
main()
{
  int i;
  rectangle r1(2,2),r2(4,6),r3(5,8);
  triangle t1(4,2),t2(5,5);
  polygon *poly_ptr[5];
  poly_ptr[0]=&r1;
  poly_ptr[1]=&t1;
  poly_ptr[2]=&r3;
  poly_ptr[3]=&t2;
  poly_ptr[4]=&r2;
  for (i=0;i<5;i++)
       poly_ptr[i] -> show();
}
```

*Με χρήση εικονικών συναρτήσεων*




*Χωρίς χρήση εικονικών συναρτήσεων*



# Εικονικές και μη εικονικές συναρτήσεις

Οι μη εικονικές συναρτήσεις επιλύονται κατά το στάδιο της μεταγλώττισης, δηλαδή η διεύθυνση του κώδικα της συνάρτησης που θα κληθεί καθίσταται γνωστή στο στάδιο της μεταγλώττισης. Ο μηχανισμός αυτός καλείται στατική σύνδεση (static binding). Αντίθετα οι εικονικές συναρτήσεις χρησιμοποιούν τον μηχανισμό της δυναμικής σύνδεσης, σύμφωνα με τον οποίο οι εικονικές συναρτήσεις επιλύονται κατά την εκτέλεση του προγράμματος.

Αυτός είναι ο λόγος για τον οποίο με τη χρήση ενός δείκτη της βασικής κλάσης μπορούμε να καλέσουμε μόνο τις συναρτήσεις-μέλη της βασικής κλάσης, ακόμη κι όταν ο δείκτης δείχνει σε αντικείμενα παράγωγων κλάσεων.

Όταν μεταγλωττίζεται ένα πρόγραμμα, ο μεταγλωττιστής δε γνωρίζει τι θα περιέχει ένας δείκτης της βασικής κλάσης, απλά γνωρίζει τον τύπο του και τον συνδέει στατικά με τις διευθύνσεις των συναρτήσεων-μελών της βασικής κλάσης στην οποία ανήκει.

# Εικονικές και μη εικονικές συναρτήσεις

**Όταν μία συνάρτηση δηλωθεί ως εικονική, η διεύθυνση της συνάρτησης που θα κληθεί καθορίζεται κατά τον χρόνο εκτέλεσης του προγράμματος και ανάλογα με την παράγωγη κλάση του αντικειμένου στο οποίο δείχνει ο δείκτης.**

**Ο μεταγλωττιστής της C++ δημιουργεί έναν πίνακα, που ονομάζεται v-table, για κάθε κλάση που περιέχει έστω και μία εικονική συνάρτηση. Κάθε ένας από αυτούς τους πίνακες περιέχει τις διευθύνσεις των εικονικών συναρτήσεων της κλάσης. Επίσης ο μεταγλωττιστής δημιουργεί ένα δείκτη (v-pointer) από κάθε αντικείμενο της κλάσης προς τον πίνακα v-table.**

**Ανάλογα λοιπόν με τον τύπο του αντικειμένου στο οποίο δείχνει ένας δείκτης της βασικής κλάσης, κατά το στάδιο της εκτέλεσης του προγράμματος, ακολουθώντας τον δείκτη v-pointer προς τον αντίστοιχο v-table της κλάσης, εντοπίζεται η διεύθυνση και επομένως ο κώδικας της συνάρτησης που θα κληθεί.**

# Γνήσιες εικονικές συναρτήσεις

**Στο προηγούμενο παράδειγμα η κλάση polygon χρησιμοποιείται αποκλειστικά ως βασική κλάση, χωρίς να παράγει δικά της αντικείμενα. Κατά συνέπεια δεν έχει νόημα οι εικονικές συναρτήσεις της να έχουν κώδικα, γιατί αυτός δεν πρόκειται ποτέ να χρησιμοποιηθεί. Για το λόγο αυτό εισάγεται η έννοια της γνήσιας εικονικής συνάρτησης (pure virtual function), η οποία δηλώνεται με την ακόλουθη γενική σύνταξη:**

```
virtual data_type function_name(parameters,…) = 0;
```

**Επομένως η γνήσια εικονική συνάρτηση είναι μία κανονική εικονική συνάρτηση χωρίς όμως σώμα. Η ανάθεση του μηδενός είναι ο τρόπος με τον οποίο ο μεταγλωττιστής αντιλαμβάνεται ότι μία εικονική συνάρτηση είναι γνήσια.**

# Γνήσιες εικονικές συναρτήσεις

Με βάση τα παραπάνω, η κλάση **polygon** του προηγούμενου παραδείγματος συντάσσεται ως ακολούθως:

```
class polygon  {
protected:
        float width,height;
public:
        void set(float a, float b)  {
                width = a;          height = b;
        }
        virtual void show() = 0;
        virtual area() = 0;
};
```

Δημιουργείται και μία συνάρτηση μέλος **area** ως γνήσια εικονική, ώστε δείκτες της βασικής κλάσης να μπορούν να χειριστούν και τη συνάρτηση **area** των παράγωγων κλάσεων.

*theory_6_vf_3.cpp*

# Γνήσιες εικονικές συναρτήσεις

```
main()  {
  int i;
  rectangle r1(2,2),r2(4,6),r3(5,8);
  triangle t1(4,2),t2(5,5);
  polygon *poly_ptr[5];
  poly_ptr[0]=&r1;
  poly_ptr[1]=&t1;
  poly_ptr[2]=&r3;
  poly_ptr[3]=&t2;
  poly_ptr[4]=&r2;
  for (i=0;i<5;i++)  {
     poly_ptr[i]->show();
     cout << "\tArea=" << poly_ptr[i]->area() << endl;
  }
}
```

```
C:\temp\try.exe
Rectangle:      2 x 2     Area=4
Triangle:       4 x 2     Area=4
Rectangle:      5 x 8     Area=40
Triangle:       5 x 5     Area=12.5
Rectangle:      4 x 6     Area=24
```

# Αφηρημένες κλάσεις

- Μία κλάση που περιέχει τουλάχιστον μία γνήσια εικονική συνάρτηση ονομάζεται αφηρημένη κλάση (abstract class). Μία αφηρημένη κλάση μπορεί να χρησιμοποιηθεί αποκλειστικά ως βασική κλάση για την παραγωγή άλλων κλάσεων. Δεν μπορούν να δημιουργηθούν αντικείμενα μίας αφηρημένης κλάσης καθώς δεν υπάρχει πλήρης ορισμός κλάσης. Μία αφηρημένη κλάση έχει μερικό ορισμό κλάσης, διότι μπορεί να περιέχει ως μέλη συναρτήσεις που δεν είναι γνήσιες εικονικές συναρτήσεις.

- Εάν παραχθεί μία κλάση από αφηρημένη κλάση, η παραγόμενη κλάση είναι και η ίδια αφηρημένη, εκτός εάν γραφούν ορισμοί για όλες τις κληρονομούμενες γνήσιες εικονικές συναρτήσεις και παράλληλα δεν εισαχθούν νέες γνήσιες εικονικές συναρτήσεις.

# Μελέτη περίπτωσης εικονικών συναρτήσεων

Στηριζόμενοι σε προηγούμενο παράδειγμα ιεραρχίας κληρονομικότητας (Point, Circle, Cylinder), θεωρούμε ότι η αρχική βασική κλάση είναι η Shape. Παρατίθενται οι λειτουργίες της κάθε κλάσης:

```
#include <iostream>
#include <string>  // C++ standard string class
#include <iomanip>
#include <vector>
using namespace std;
class Shape  {
public:
    virtual double getArea()  {
        return 0.0;
    }
```

*theory_6_case_study_1.cpp*

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
    virtual double getVolume()
    {
        return 0.0;
    }
    virtual string getName() = 0; // επιστρέφει το όνομα της Shape
    virtual void print() = 0;        // έξοδος της Shape
};  // τέλος της κλάσης Shape

class Point : public Shape
{
private:
    int x;
    int y;
```

*Εικονική συνάρτηση*

*Γνήσιες εικονικές συναρτήσεις*

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
public:
   Point(int xValue, int yValue)  {
      x = xValue;
      y = yValue;
   }
   void setX(int xValue)  {
      x = xValue;
   }
   int getX() const  {
      return x;
   }
   void setY(int yValue)  {
      y = yValue;
   }
```

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
    int getY()
    {
        return y;
    }
    string getName()
    {
        return "Point";
    }
    void print()
    {
        cout << '[' << getX() << ", " << getY() << ']';
    }
};
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
class Circle : public Point
{
private:
   double radius;
public:
   Circle( int xValue, int yValue, double radiusValue )
            : Point( xValue, yValue )  // call base-class constructor
   {
      setRadius(radiusValue);
   }
   void setRadius(double radiusValue)  {
      radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
   }
   double getRadius()  {
      return radius;
}
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
double getDiameter()  {
    return 2 * getRadius();
}
double getCircumference()  {
    return 3.14159 * getDiameter();
}
double getArea()  {
    return 3.14159 * getRadius() * getRadius();
}
// υπέρβαση της εικονικής συνάρτησης getName:
// επιστρέφεται το όνομα της Circle
string getName()  {
    return "Circle";
}
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
   // υπέρβαση της εικονικής συνάρτησης print:
   void print()
   {
      cout << "center is ";
      Point::print();  // κλήση της συνάρτησης print της Point
      cout << "; radius is " << getRadius();
   }
}; // τέλος της κλάσης Circle

class Cylinder : public Circle
{
private:
   double height;
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
public:
   Cylinder(int xValue, int yValue, double radiusValue, double
                  heightValue) : Circle(xValue, yValue, radiusValue)
   {
      setHeight(heightValue);
   }
   void setHeight(double heightValue)
   {
      height = ( heightValue < 0.0 ? 0.0 : heightValue );
   }
   double getHeight()
   {
      return height;
   }
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
// υπέρβαση της εικονικής συνάρτησης getArea:
// επιστρέφεται το εμβαδόν της Cylinder
double Cylinder::getArea()
{
   return 2 * Circle::getArea() + getCircumference()
                                          * getHeight();
}
// υπέρβαση της εικονικής συνάρτησης getVolume:
// επιστρέφεται ο όγκος της Cylinder
double Cylinder::getVolume()
{
   return Circle::getArea() * getHeight();
}
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
   virtual string getName()  {
      return "Cylinder";
   }
   virtual void print()  {
      Circle::print();  // επαναχρησιμοποίηση κώδικα
      cout << "; height is " << getHeight();
   }
};  // τέλος της Cylinder
// Κλήση εικονικών συναρτήσεων μέσω δείκτη στη βασική κλάση
// με χρήση δυναμικής σύνδεσης (ισοδύναμη με την επόμενη συνάρτηση)
void virtualViaPointer(Shape *baseClassPtr)  {
   cout << baseClassPtr->getName() << ": ";
   baseClassPtr->print();
   cout << "\narea is " << baseClassPtr->getArea()
        << "\nvolume is " << baseClassPtr->getVolume() << "\n\n";
}
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
// Κλήση εικονικών συναρτήσεων κατ' αναφορά της
// βασικής κλάσης με χρήση δυναμικής σύνδεσης
// (ισοδύναμη με την προηγούμενη συνάρτηση)
void virtualViaReference(Shape &baseClassRef)  {
   cout << baseClassRef.getName() << ": ";
   baseClassRef.print();
   cout << "\narea is " << baseClassRef.getArea()
         << "\nvolume is " << baseClassRef.getVolume() << "\n\n";
}
main()  {
   int i,j;
   cout << fixed << setprecision(2);
   Point point( 7, 11 );
   Circle circle( 22, 8, 3.5 );
   Cylinder cylinder( 10, 10, 3.3, 10 );
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
cout << point.getName() << ": ";    // στατική σύνδεση
point.print();                      // στατική σύνδεση
cout << '\n';
cout << circle.getName() << ": ";   // στατική σύνδεση
circle.print();                     // στατική σύνδεση
cout << '\n';
cout << cylinder.getName() << ": "; // στατική σύνδεση
cylinder.print();                   // στατική σύνδεση
cout << "\n\n";
vector < Shape * > shapeVector(3);
// ο shapeVector[0] δείχνει σε αντικείμενο της παραγόμενης
// κλάσης Point
shapeVector[ 0 ] = &point;
// ο shapeVector[1] δείχνει σε αντικείμενο της παραγόμενης
// κλάσης Circle
shapeVector[ 1 ] = &circle;
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
// ο shapeVector[2] δείχνει σε αντικείμενο της παραγόμενης
// κλάσης cylinder
shapeVector[ 2 ] = &cylinder;
// χρήση δυναμικής σύνδεσης
cout << "\nVirtual function calls made off
     << "base-class pointers:\n\n";
for (i=0;i<shapeVector.size();i++)
   virtualViaPointer( shapeVector[ i ] );
cout < "\nVirtual function calls made off
     << "base-class references:\n\n";
for (j=0;j<shapeVector.size();j++)
   virtualViaReference( *shapeVector[ j ] );
}
```

# Μελέτη περίπτωσης εικονικών συναρτήσεων

```
C:\temp\try.exe
Point: [7, 11]
Circle: center is [22, 8]; radius is 3.50
Cylinder: center is [10, 10]; radius is 3.30; height is 10.00


Virtual function calls made off base-class pointers:

Point: [7, 11]
area is 0.00
volume is 0.00

Circle: center is [22, 8]; radius is 3.50
area is 38.48
volume is 0.00

Cylinder: center is [10, 10]; radius is 3.30; height is 10.00
area is 275.77
volume is 342.12


Virtual function calls made off base-class references:

Point: [7, 11]
area is 0.00
volume is 0.00

Circle: center is [22, 8]; radius is 3.50
area is 38.48
volume is 0.00

Cylinder: center is [10, 10]; radius is 3.30; height is 10.00
area is 275.77
volume is 342.12
```

# Εικονικές συναρτήσεις αποδόμησης

- **Στην περίπτωση που χρησιμοποιείται δείκτης βασικής κλάσης σε αντικείμενο της παραγόμενης κλάσης, εάν καταστρέψουμε το αντικείμενο με την delete η συμπεριφορά θα είναι απροσδιόριστη.**

- **Ως λύση δίνονται οι εικονικές συναρτήσεις αποδόμησης (virtual destructors): Δηλώνεται η συνάρτηση αποδόμησης της βασικής κλάσης ως εικονική, οπότε όταν καλείται η delete καλείται και η κατάλληλη συνάρτηση αποδόμησης.**

- **Όταν καταστρέφουμε ένα αντικείμενο μιας παραγόμενης κλάσης πρώτα εκτελείται η συνάρτηση αποδόμησης της παράγωγης κλάσης και μετά εκτελείται η συνάρτηση αποδόμησης της βασικής κλάσης.**

# Μελέτη περίπτωσης πολυμορφισμού

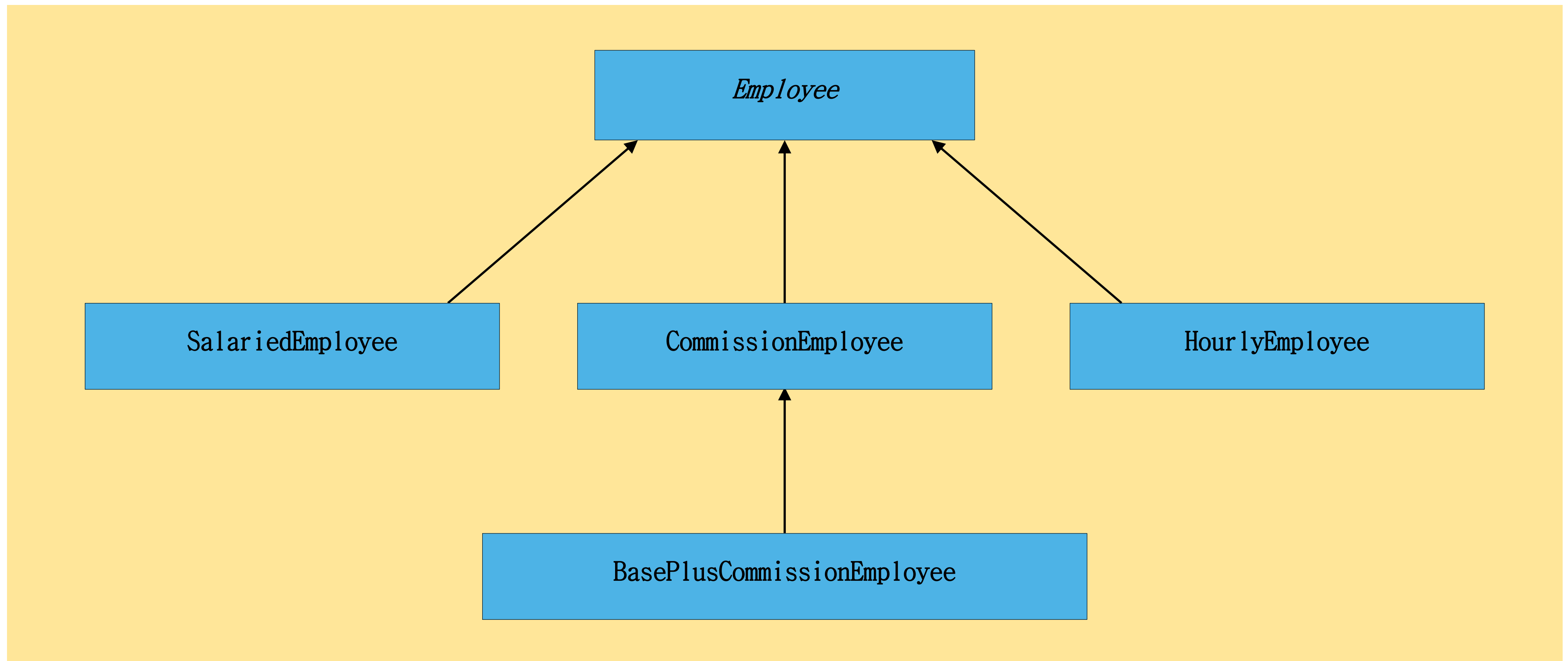
- Πρόγραμμα μισθοδοσίας, όπου γίνεται χρήση εικονικών συναρτήσεων και πολυμορφισμού.

- **Πρόβλημα:** 4 είδη εργαζομένων, οι οποίοι αμείβονται σε εβδομαδιαία βάση
  - Salaried (μισθός, ανεξαρτήτα από τις ώρες εργασίας)
  - Hourly (υπερωρίες [>40 ώρες] + 50% το ωρομίσθιο)
  - Commission (ποσοστό επί των πωλήσεων)
  - Base-plus-commission (βασικός μισθός + ποσοστό επί των πωλήσεων)

*Deitel & Deitel: C++ Programming*

# Μελέτη περίπτωσης πολυμορφισμού

- Βασική κλάση: **Employee**, η οποία είναι γνήσια εικονική συνάρτηση.
- Παραγόμενες κλάσεις από την Employee:



# Μελέτη περίπτωσης πολυμορφισμού

```
1
2   // Βασική κλάση Employee
3   #ifndef EMPLOYEE_H
4   #define EMPLOYEE_H
5
6   #include <string>  // C++ πρότυπη κλάση string
7
8   using std::string;
9
10  class Employee {
11
12  public:
13     Employee(string &, string &, string &);
14
15     void setFirstName(string &);
16     string getFirstName();
17
18     void setLastName(string &);
19     string getLastName();
20
21     void setSocialSecurityNumber(string & );
22     string getSocialSecurityNumber();
23
```

# Μελέτη περίπτωσης πολυμορφισμού

```
24     // Η γνήσια εικονική συνάρτηση καθιστά την Employee βασική κλάση
25     virtual double earnings() = 0;  // γνήσια εικονική
26     virtual void print();           // εικονική
27
28  private:
29     string firstName;
30     string lastName;
31     string socialSecurityNumber;
32
33  };
34
35  #endif // EMPLOYEE_H
```

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // employee.cpp
2   // Ορισμοί των συναρτήσεων-μελών της βασικής κλάσης Employee
3   // ΣΗΜΕΙΩΣΗ: Δεν παρέχονται ορισμοί για γνήσιες εικονικές συναρτήσεις
4   #include <iostream>
5
6   using std::cout;
7   using std::endl;
8
9   #include "employee.h"  // Ορισμός της κλάσης Employee
10
11  // Συνάρτηση δόμησης
12  Employee::Employee(string &first, string &last,
13     string &SSN )
14     : firstName( first ),
15       lastName( last ),
16       socialSecurityNumber( SSN )
17  {
18     // άδειο σώμα συνάρτησης
19
20  }
21
```

```
22  // Επιστρέφεται το όνομα
23  string Employee::getFirstName()
24  {
25     return firstName;
26
27  }
28
29  // Επιστρέφεται το επώνυμο
30  string Employee::getLastName()
31  {
32     return lastName;
33
34  }
35
36  // Επιστρέφεται ο αριθμός κοινωνικής ασφάλισης
37  string Employee::getSocialSecurityNumber()
38  {
39     return socialSecurityNumber;
40
41  }
42
43
44  void Employee::setFirstName(string &first)
45  {
46     firstName = first;
47
48  }
49
```

# Μελέτη περίπτωσης πολυμορφισμού

```
50
51  void Employee::setLastName(string &last)
52  {
53     lastName = last;
54
55  }
56
57
58  void Employee::setSocialSecurityNumber(string &number)
59  {
60     socialSecurityNumber = number;
61
62  }
63
64
65  void Employee::print()
66  {
67     cout << getFirstName() << ' ' << getLastName()
68          << "\nsocial security number: "
69          << getSocialSecurityNumber() << endl;
70
71  }
```

**Πρότυπη υλοποίηση για την εικονική συνάρτηση *print*.**

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // salaried.h
2   // Η κλάση SalariedEmployee class απορρέει από την Employee.
3   #ifndef SALARIED_H
4   #define SALARIED_H
5
6   #include "employee.h"
7
8   class SalariedEmployee : public Employee {
9
10  public:
11     SalariedEmployee(string &, string &,
12        string &, double = 0.0 );
13
14     void setWeeklySalary( double );
15     double getWeeklySalary();
16
17     virtual double earnings();
18     virtual void print();
19
20  private:
21     double weeklySalary;
22
23  };
24
25  #endif
```

Νέες συναρτήσεις για την κλάση ***SalariedEmployee***.

Η ***earnings*** υπερβαίνεται. Η ***print*** υπερβαίνεται για να καθορισθεί ότι το αντικείμενο είναι ***salaried employee***.

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // salaried.cpp
2   // Ορισμοί των συναρτήσεων-μέλος της κλάσης SalariedEmployee
3   #include <iostream>
4
5   using std::cout;
6
7   #include "salaried.h"
8
9   // Συνάρτηση δόμησης της κλάσης SalariedEmployee
10  SalariedEmployee::SalariedEmployee(string &fir
11     string &last, string &socialSecurityNumber,
12     double salary )
13     : Employee( first, last, socialSecurityNumb
14  {
15     setWeeklySalary( salary );
16
17  }
18
19
20  void SalariedEmployee::setWeeklySalary( double salary )
21  {
22     weeklySalary = salary < 0.0 ? 0.0 : salary;
23
24  }
25
```

**Χρήση συναρτήσεων δόμησης της βασικής κλάσης για τα βασικά πεδία.**

# Μελέτη περίπτωσης πολυμορφισμού

```
26
27  double SalariedEmployee::earnings()
28  {
29     return getWeeklySalary();
30
31  }
32
33
34  double SalariedEmployee::getWeeklySalary()
35  {
36     return weeklySalary;
37
38  }
39
40
41  void SalariedEmployee::print()
42  {
43     cout << "\nsalaried employee: ";
44     Employee::print();  // επαναχρησιμοποίηση κώδικα
45
46  }
```

Υλοποίηση γνήσιων εικονικών συναρτήσεων.

```
1  // hourly.h
2
3  #ifndef HOURLY_H
4  #define HOURLY_H
5
6  #include "employee.h"
7
8  class HourlyEmployee : public Employee {
9
10 public:
11    HourlyEmployee( string &, string &,
12       string &, double = 0.0, double = 0.0 );
13
14    void setWage( double );
15    double getWage() const;
16
17    void setHours( double );
18    double getHours();
19
20    virtual double earnings();
21    virtual void print();
22
23 private:
24    double wage;   // ημερομίσθιο
25    double hours;  // εβδομαδιαίο ωράριο
26
27 };
28
29 #endif
```

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // hourly.cpp
2   // Ορισμοί των συναρτήσεων-μέλος της κλάσης HourlyEmployee
3   #include <iostream>
4
5   using std::cout;
6
7   #include "hourly.h"
8
9   // Συνάρτηση δόμησης της κλάσης HourlyEmployee
10  HourlyEmployee::HourlyEmployee( string &first,
11     string &last, string &socialSecurityNumber,
12     double hourlyWage, double hoursWorked )
13     : Employee( first, last, socialSecurityNumber )
14  {
15     setWage( hourlyWage );
16     setHours( hoursWorked );
17
18  }
19
20
21  void HourlyEmployee::setWage( double wageAmount )
22  {
23     wage = wageAmount < 0.0 ? 0.0 : wageAmount;
24
25  }
```

# Μελέτη περίπτωσης πολυμορφισμού

```
26
27
28  void HourlyEmployee::setHours( double hoursWorked )
29  {
30     hours = ( hoursWorked >= 0.0 && hoursWorked <= 168.0 ) ?
31        hoursWorked : 0.0;
32
33  }
34
35
36  double HourlyEmployee::getHours()
37  {
38     return hours;
39
40  }
41
42
43  double HourlyEmployee::getWage()
44  {
45     return wage;
46
47  }
48
```

# Μελέτη περίπτωσης πολυμορφισμού

```
49
50  double HourlyEmployee::earnings()
51  {
52     if ( hours <= 40 )  // δεν υπάρχουν υπερωρίες
53        return wage * hours;
54     else                // η υπερωρία αμείβεται 150% του ημερομίσθιου
55        return 40 * wage + ( hours - 40 ) * wage * 1.5;
56
57  }
58
59
60  void HourlyEmployee::print()
61  {
62     cout << "\nhourly employee: ";
63     Employee::print();  // επαναχρησιμοποίηση κώδικα
64
65  }
```

```
1   // commission.h
2
3   #ifndef COMMISSION_H
4   #define COMMISSION_H
5
6   #include "employee.h"
7
8   class CommissionEmployee : public Employee {
9
10  public:
11     CommissionEmployee(string &, string &,
12         string &, double = 0.0, double = 0.0 );
13
14     void setCommissionRate( double );
15     double getCommissionRate();
16
17     void setGrossSales( double );
18     double getGrossSales();
19
20     virtual double earnings();
21     virtual void print();
22
23  private:
24     double grossSales;      // εβδομαδιαίος τζίρος
25     double commissionRate;  // ποσοστό προμήθειας
26
27  };
28
29  #endif
```

Καθορισμός ***rate*** και ***sales***.

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // commission.cpp
2   // Ορισμοί των συναρτήσεων-μέλος της κλάσης CommissionEmployee
3   #include <iostream>
4
5   using std::cout;
6
7   #include "commission.h"
8
9   // Συνάρτηση δόμησης της κλάσης CommissionEmployee
10  CommissionEmployee::CommissionEmployee(string &first,
11     string &last, string &socialSecurityNumber,
12     double grossWeeklySales, double percent )
13     : Employee( first, last, socialSecurityNumber )
14  {
15     setGrossSales( grossWeeklySales );
16     setCommissionRate( percent );
17
18  }
19
20
21  double CommissionEmployee::getCommissionRate()
22  {
23      return commissionRate;
24
25  }
```

# Μελέτη περίπτωσης πολυμορφισμού

```
26
27
28 double CommissionEmployee::getGrossSales()
29 {
30     return grossSales;
31
32 } //
33
34
35 void CommissionEmployee::setGrossSales( double sales )
36 {
37    grossSales = sales < 0.0 ? 0.0 : sales;
38
39 }
40
41
42 void CommissionEmployee::setCommissionRate( double rate )
43 {
44     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
45
46 }
47
```

# Μελέτη περίπτωσης πολυμορφισμού

```
48
49  double CommissionEmployee::earnings()
50  {
51     return getCommissionRate() * getGrossSales();
52
53  }
54
55
56  void CommissionEmployee::print()
57  {
58     cout << "\ncommission employee: ";
59     Employee::print();  // επαναχρησιμοποίηση κώδικα
60
61  }
```

ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // baseplus.h
2   // Η κλάση BasePlusCommissionEmployee απορρέει από την κλάση Employ
3   #ifndef BASEPLUS_H
4   #define BASEPLUS_H
5
6   #include "commission.h"
7
8   class BasePlusCommissionEmployee : public CommissionEmployee {
9
10  public:
11     BasePlusCommissionEmployee(string &, string &,
12        string &, double = 0.0, double = 0.0, double = 0.0 );
13
14     void setBaseSalary( double );
15     double getBaseSalary();
16
17     virtual double earnings();
18     virtual void print();
19
20  private:
21     double baseSalary;       // βασικός εβδομαδιαίος μισθός
22
23  };
24
25  #endif
```

Κληρονομεί από την κλάση ***CommissionEmployee*** (και έμμεσα από την κλάση ***Employee***).

```
1   // baseplus.cpp
2   // Ορισμοί των συναρτήσεων-μέλος της κλάσης BasePlusCommissionEmployee
3   #include <iostream>
4
5   using std::cout;
6
7   #include "baseplus.h"
8
9   // Συνάρτηση δόμησης για την κλάση BasePlusCommissionEmployee
10  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     string &first, string &last,
12     const string &socialSecurityNumber,
13     double grossSalesAmount, double rate,
14     double baseSalaryAmount )
15     : CommissionEmployee( first, last, socialSecurityNumber,
16       grossSalesAmount, rate )
17  {
18     setBaseSalary( baseSalaryAmount );
19
20  }
21
22
23  void BasePlusCommissionEmployee::setBaseSalary( double salary )
24  {
25     baseSalary = salary < 0.0 ? 0.0 : salary;
26
27  }
```

# Μελέτη περίπτωσης πολυμορφισμού

```
28
29
30  double BasePlusCommissionEmployee::getBaseSalary()
31  {
32      return baseSalary;
33
34  }
35
36
37  double BasePlusCommissionEmployee::earnings()
38  {
39      return getBaseSalary() + CommissionEmployee::earnings();
40
41  }
42
43
44  void BasePlusCommissionEmployee::print()
45  {
46     cout << "\nbase-salaried commission employee: ";
47     Employee::print();  // επαναχρησιμοποίηση κώδικα
48
49  }
```

# Μελέτη περίπτωσης πολυμορφισμού

```
1   // fig10_33.cpp
2   // Ιεραρχία της κλάσης Employee
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7   using std::fixed;
8
9   #include <iomanip>
10
11  using std::setprecision;
12
13  #include <vector>
14
15  using std::vector;
16
17  #include <typeinfo>
18
19  #include "employee.h"    // Βασική κλάση Employee
20  #include "salaried.h"    // Κλάση SalariedEmployee
21  #include "commission.h"  // Κλάση CommissionEmployee
22  #include "baseplus.h"    // Κλάση BasePlusCommissionEmployee
23  #include "hourly.h"      // Κλάση HourlyEmployee
24
```

# Μελέτη περίπτωσης πολυμορφισμού

```
25 int main()
26 {
27
28    cout << fixed << setprecision( 2 );
29
30
31    vector < Employee * > employees( 4 );
32
33
34    employees[ 0 ] = new SalariedEmployee( "John", "Smith",
35       "111-11-1111", 800.00 );
36    employees[ 1 ] = new CommissionEmployee( "Sue", "Jones",
37       "222-22-2222", 10000, .06 );
38    employees[ 2 ] = new BasePlusCommissionEmployee( "Bob",
39       "Lewis", "333-33-3333", 300, 5000, .04 );
40    employees[ 3 ] = new HourlyEmployee( "Karen", "Price",
41       "444-44-4444", 16.75, 40 );
42
```

```
43
44    for ( int i = 0; i < employees.size(); i++ ) {
45
46
47       employees[ i ]->print();
48
49
50       BasePlusCommissionEmployee *commissionPtr =
51          dynamic_cast < BasePlusCommissionEmployee * >
52             ( employees[ i ] );
53
54
55
56       if ( commissionPtr != 0 ) {
57          cout << "old base salary: $"
58               << commissionPtr->getBaseSalary() << endl;
59          commissionPtr->setBaseSalary(
60             1.10 * commissionPtr->getBaseSalary() );
61          cout << "new base salary with 10% increase is: $"
62               << commissionPtr->getBaseSalary() << endl;
63
64       }
65
66       cout << "earned $" << employees[ i ]->earnings() << endl;
67
68    }
69
```

Χρήση *downcasting*. Εάν «δείχνει» στον σωστό τύπο αντικειμένου, ο δείκτης είναι μη μηδενικός. Έτσι δίνεται αύξηση μόνο σε ***BasePlusCommissionEmployees***.

# Μελέτη περίπτωσης πολυμορφισμού

```
70     // απελευθέρωση μνήμης
71     for ( int j = 0; j < employees.size(); j++ ) {
72
73
74        cout << "\ndeleting object of "
75               << typeid( *employees[ j ] ).name();
76
77        delete employees[ j ];
78
79     }
80
81     cout << endl;
82
83     return 0;
84
85  } // τέλος της main
```

Η ***typeid*** επιστρέφει ένα αντικείμενο τύπου ***type_info***, το οποίο περιέχει πληροφορίες για τον τελεστή, συμπεριλαμβανομένου του ονόματός του.

# Μελέτη περίπτωσης πολυμορφισμού

```
salaried employee: John Smith
social security number: 111-11-1111
earned $800.00

commission employee: Sue Jones
social security number: 222-22-2222
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
old base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

hourly employee: Karen Price
social security number: 444-44-4444
earned $670.00

deleting object of class SalariedEmployee
deleting object of class CommissionEmployee
deleting object of class BasePlusCommissionEmployee
deleting object of class HourlyEmployee
```

# Τέλος Ενότητας