Πανεπιστήμιο Αιγαίου

Μεθοδολογίες και Γλώσσες Προγραμματισμού Ι

C++ summary

Εργίνα Καβαλλιεράτου (kavallieratou@aegean.gr)

Μόνιμη Επίκουρος Καθηγήτρια

Τμήμα Μηχανικών Πληροφοριακών & Επικοινωνιακών Συστημάτων

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

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.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» 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.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΠΡΟΣΟΧΗ!

- ✓ Αυτή η σύνοψη, περιλαμβάνει κυρίως ύλη της C++. Οι γνώσεις από C θεωρούνται δεδομένες
- ✓ Το μάθημα «Αντικειμενοστρεφής Ανάλυση» πρέπει να διαβαστεί προσεκτικά στο σύνολο του

Include Files

- ✓ Η εντολή **#include** εισάγει το αρχείο που υποδεικνύει στον κώδικα του προγράμματος
- ✓ Πρόκειται συνήθως για *header files*, που περιέχουν ορισμούς συναρτήσεων δεδομένων, κλάσεων κλπ.
- ✓ Έχει δύο εκδοχές:
  1) **#include <filename>**
  2) **#include "filename"**

Κλάση/Αντικείμενο

- ✓ Στον ΑΠ δεν ασχολούμαστε με το πώς θα χωρίσουμε το πρόβλημα σε συναρτήσεις αλλά σε κλάσεις και αντικείμενα.
- ✓ Μία κλάση μπορεί να έχει ένα ή περισσότερα αντικείμενα.
- ✓ Κάθε αντικείμενο μπορεί να περιέχει μία ή περισσότερες συναρτήσεις, **συναρτήσεις-μέλη**
- ✓ Η κλάση λειτουργεί ως πρότυπο: δηλώνουμε σε αυτή δεδομένα και συναρτήσεις που θα έχουν τα αντικείμενα της
- ✓ Όταν δημιουργήσουμε αντικείμενα θα έχουν ότι έχει η κλάση που ανήκουν

C++ Class

```
class Classic_Example {
public:
// Δεδομένα και διαδικασίες προσβάσιμα από παντού
protected:
//Δεδομένα και διαδικασίες προσβάσιμα από την κλάση
//  τις παραγόμενες κλάσεις και  friends κλάσεις
private:
// Δεδομένα και διαδικασίες προσβάσιμα από την κλάση
// και  friends κλάσεις
};
```

Επικοινωνία με την κλάση

- ✓ Για να καλέσουμε μία συνάρτηση-μέλος από το κυρίως πρόβλημα γίνεται ΠΑΝΤΑ με αναφορά στο αντικείμενο που αντιστοιχεί:

<όνομα αντικειμένου>.<όνομα συνάρτησης-μέλους>

Πρόσβαση σε στοιχεία κλάσης

- ✓ Αν δεν διευκρινιστεί ο τύπος των **δεδομένων** στην κλάση, ο εξ' ορισμού τύπος είναι private
- ✓ Στα private δεδομένα δεν μπορούμε να έχουμε πρόσβαση εκτός κλάσης.
- ✓ Αυτό συμβαίνει για την ασφάλεια των δεδομένων
- ✓ Για το λόγο αυτό πρέπει να δημιουργήσουμε κατάλληλες συναρτήσεις για εργασίες όπως
  - Απόδοση τιμής σε μεταβλητή
  - Επιστροφή δεδομένων κλπ

Τελεστής Εμβέλειας ::

‘Όταν δηλώνουμε πρότυπο συνάρτησης μέσα σε μία κλάση και θέλουμε να ορίσουμε τη συνάρτηση εκτός κλάσης πρέπει να χρησιμοποιήσουμε τον Τελεστής Εμβέλειας ::

Απόδοση τιμής σε μεταβλητή

- ✓ Υπάρχουν συναρτήσεις που χρησιμοποιούνται για να αποδώσουν τιμή σε private μεταβλητή μιας κλάσης
- ✓ Συνήθως ονομάζονται

  Set_<όνομα μεταβλητής>

Επιστροφή δεδομένων

- ✓ Υπάρχουν συναρτήσεις που χρησιμοποιούνται για να μεταφέρουν τιμή private μεταβλητής εκτός κλάσης
- ✓ Συνήθως ονομάζονται

  Get_<όνομα μεταβλητής>

Αρχικοποίηση κλάσης

- ✓ Οι classes περιλαμβάνουν μια ιδιαίτερη συνάρτηση-μέλος που ονομάζεται constructor (δομητής).
- ✓ Ο constructor έχει το ίδιο όνομα με την κλάση
- ✓ Μπορεί να παίρνει παράμετρο αν χρειάζεται αλλά δεν έχει τύπο επιστροφής, ούτε καν void

Constructor (Δομητής)

```
class Date
{
public:
        Date(int d, int m, int y);
        // . . .
};

Date::Date(int d, int m, int y)
: day(d),
  month(m),
  year(y)
{ }
```

Constructor (Δομητής)

- ✓ Μεγάλο πλεονέκτημα των ΔΟΜΗΤΩΝ είναι το ότι η ύπαρξή τους απομακρύνει την πιθανότητα μη αρχικοποιημένων μεταβλητών. Αν έχει δηλωθεί η Date(int, int, int)

  τότε η

  ```
  Date d;
  ```

  είναι λάθος

- ✓ Στις κλάσεις μπορεί να ορίζονται περισσότεροι από ένας δομητές αλλά μόνο ένας από αυτούς θα είναι ο προκαθορισμένος (default)

Constructor (Δομητής)

```
class Date
{
public:
        Date();
        Date(int d, int m, int y);
        Date(String);
        // . . .
};

Date::Date(int d, int m, int y)
: day(d),
  month(m),
  year(y)
{ if (year == 0) year = current year;}
```

Destructor (αποδομητής)

- ✓ Όταν δημιουργούμε ένα constructor, δημιουργούμε και ένα Destructor (αποδομητής)
- ✓ Δεν έχει παραμέτρους και δεν επιστρέφει τίποτα
- ✓ Πρέπει να αποδεσμεύσει τη δεσμευμένη μνήμη memory και να καθαρίσει την κλάση
- ✓ Ο destructor έχει τη μορφή:

~ClassName() {…}

Constructor και destructor

- ✓ Αν δεν δηλώσουμε constructor ή/και destructor ο compiler χρησιμοποιεί προκαθορισμένους.
- ✓ Οι προκαθορισμένοι constructor και destructor δεν κάνουν τίποτα και δεν επιστρέφουν τίποτα, ισοδυναμούν με:

cat() & ~cat()

- ✓ Όταν δηλώσουμε ένα αντικείμενο κλάσης καλούμε τον constructor
- ✓ Αν ο constructor δεν παίρνει παραμέτρους, μπορούμε να γράψουμε:

Cat Frisky;

- ✓ Συνίσταται όταν δηλώνουμε constructor να δηλώνουμε και destructor

Υπερφόρτωση συναρτήσεων

- ✓ Η C++ επιτρέπει τη πολλαπλή δημιουργία συναρτήσεων με το ίδιο όνομα
- ✓ Οι συναρτήσεις αυτές πρέπει να διαφέρουν στο πλήθος ή τον τύπο των παραμέτρων:

  int myFunction (int, int);

  int myFunction (long, long);

  int myFunction (long);

- ✓ Κάθε φορά καλείται αυτόματα η κατάλληλη ανάλογα με τις παραμέτρους που περνάμε.

Αναδρομή

- ✓Έχουμε αναδρομή όταν μία συνάρτηση καλεί τον εαυτό της
- ✓Η αναδρομή είναι πολύ χρήσιμη σε περίπτωση πολύπλοκων υπολογισμών

Σχεδιασμός αναδρομικών συναρτήσεων

- ✓ Ορίζουμε συνθήκη τερματισμού:
  - Πρόκειται για την περίπτωση όπου η συνάρτηση παύει να καλεί τον εαυτό της
- ✓ Ορίζουμε επανάκληση της συνάρτησης από τον εαυτό της

συνθήκη τερματισμού

- ✓ Η συνθήκη τερματισμού αντιστοιχεί σε μια περίπτωση που γνωρίζουμε ήδη την απάντηση ή υπολογίζεται εύκολα
- ✓ Αν δεν έχουμε συνθήκη τερματισμού δεν πρέπει να χρησιμοποιήσουμε αναδρομή ή δεν έχουμε καταλάβει το πρόβλημα

Επανάκληση

- ✓ Η επανάκληση χρησιμοποιείται για να λύσουμε κάποιο υπο-πρόβλημα
  - Σε κάθε επανάκληση οι παράμετροι θα πρέπει να διαφέρουν ώστε να πλησιάζουμε στη λύση

Συναρτήσεις-μέλη const

- ✓ Με μία συνάρτηση const μέσα στην κλάση, πληροφορούμε τον compiler, ότι δεν αλλάζει τα δεδομένα της κλάσης

```
void SomeFunction() const;
int GetAge() const;
```

- ✓ Είναι καλή προγραμματιστική τεχνική να δηλώνουμε το μέγιστο δυνατό αριθμό const συναρτήσεων για προστασία

Inline συναρτήσεις

- Αν μας ενδιαφέρει η ταχύτητα εκτέλεσης και η συνάρτηση που θέλουμε να γράψουμε είναι απλή τότε μπορούμε να τη δηλώσουμε ως inline π.χ.

```
inline int square(int x) {return x*x;}
```

Inline συναρτήσεις

- ✓ Μπορούμε να γράψουμε εκτός κλάσης

```
inline int Cat::GetWeight(){
return itsWeight;
}
```

- ✓ Ή αν τη δηλώσουμε μέσα στην κλάση γίνεται αυτόματα inline

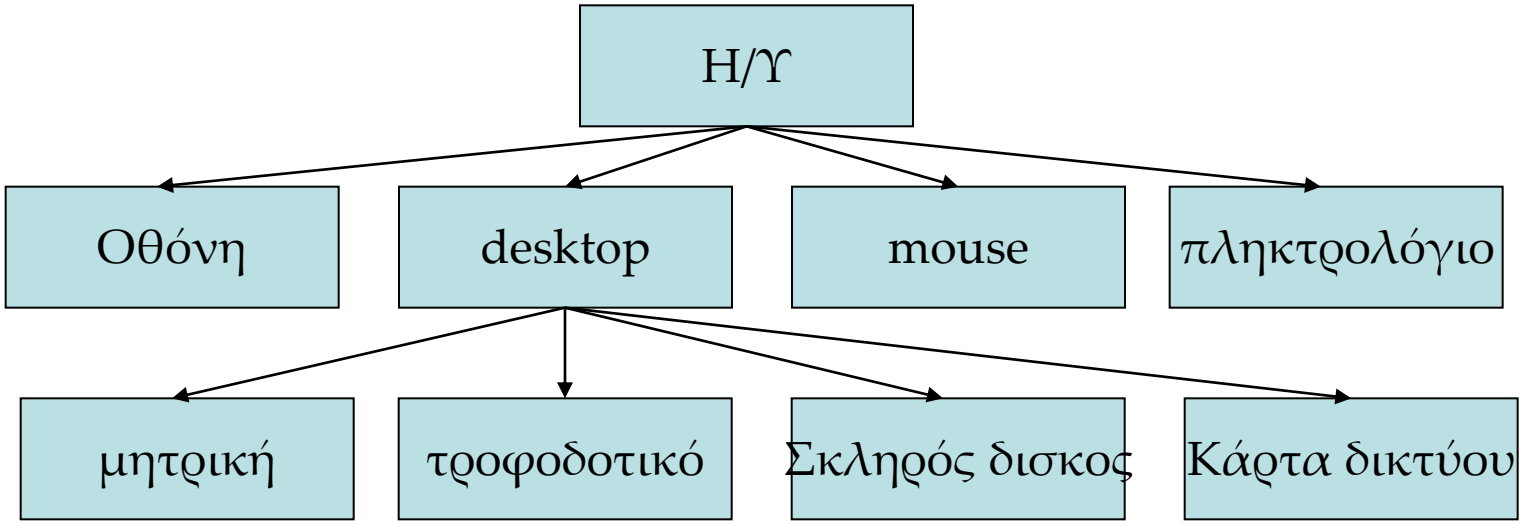
```
class Cat{
public:
int GetWeight() { return itsWeight; }    // inline
void SetWeight(int aWeight);
};
```

Δηλώσεις κλάσεων

- ✓ Συνηθίζεται να δηλώνουμε τις κλάσεις, καθώς και όλες τις δηλώσεις ενός προγράμματος σε ξεχωριστό αρχείο.
- ✓ Αυτό το αρχείο ονομάζεται header file, έχει το ίδιο όνομα με το αρχείο του προγράμματος και επέκταση .h ή .hpp
- ✓ Αν δηλώσουμε μία κλάση σε ξεχωριστό αρχείο, το αρχείο παίρνει το όνομα της κλάσης.
- ✓ Στο κυρίως αρχείο πρέπει να συμπεριλάβουμε τα header files με τις βιβλιοθήκες.

Σύνθετες κλάσεις

- ✓ Πρόκειται για κλάσεις, που στα δεδομένα τους συμπεριλαμβάνουν άλλες απλούστερες κλάσεις
- ✓ Αυτό δηλώνει μία σχέση μεταξύ των κλάσεων



Ανάγνωση από αρχείο

- Για να διαβάσουμε από αρχείο πρέπει να δηλώσουμε την αντίστοιχη κλάση:

```
#include <fstream>
```

- Καθώς και αντικείμενο της κλάσης:

```
ifstream inFile;
```

- Να ανοίξουμε το αρχείο:

```
inFile.open("C:\\temp\\datafile.txt");
```

- Να ελέγξουμε αν άνοιξε:

```
if (!inFile) {
    cerr << "Unable to open file datafile.txt";
    exit(1);   // call system to stop
}
```

- Και όταν τελειώσουμε να το κλείσουμε:

```
inFile.close();
```

Χρήση pointer

```
int *pAge = 0;
```

- ✓ Ο pAge είναι δείκτης σε ακέραιο δηλαδή θα έχει τη διεύθυνση μιας θέσης μνήμης αρκετά μεγάλης για να αποθηκεύσει ακέραιο.
- ✓ Όταν η διεύθυνση ενός δείκτη είναι 0 κατά σύμβαση ο δείκτης είναι null.
- ✓ Καλό είναι οι δείκτες να αρχικοποιούνται κατά τη δήλωση τους:

```
unsigned short int howOld = 50;
unsigned short int * pAge = &howOld;
```

- ✓ Ο δείκτης παρέχει έμμεση πρόσβαση στη μεταβλητή, της οποίας τη διεύθυνση περιέχει:

```
unsigned short int yourAge;
yourAge = *pAge;
```

Γενικοί δείκτες: void*

- ✓ Δήλωση: **void * ptr;**
- ✓ Μπορεί να δείξει σε οποιοδήποτε τύπο

  void * ptr = GetAddress();

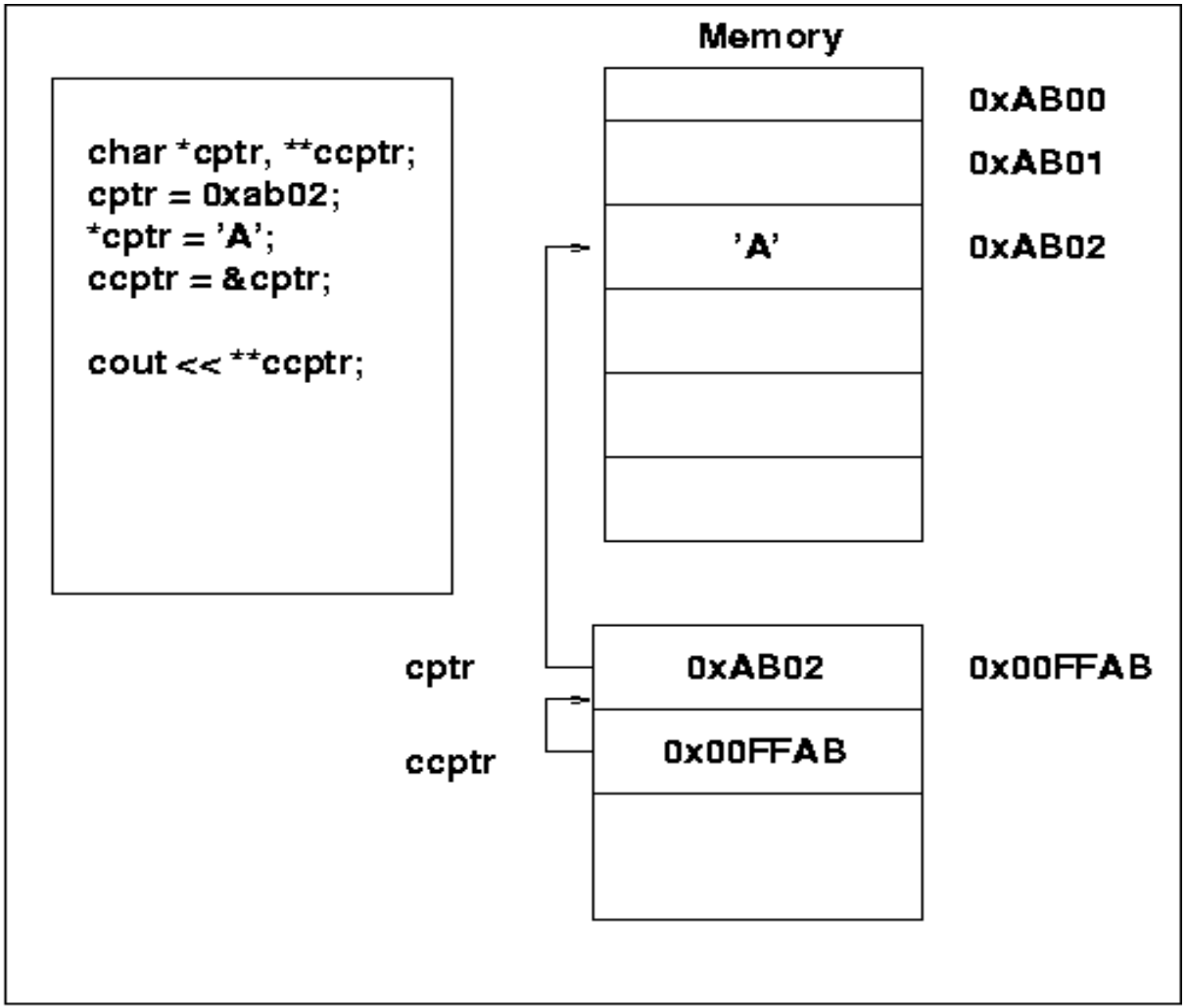
  float *fptr = (float*)ptr;
- ✓ Οι συναρτήσεις μπορούν να λάβουν και να επιστρέψουν δείκτες void
- ✓ ΟΛΟΙ οι δείκτες έχουν το ίδιο μέγεθος

Ανάκληση

- ✓ Μπορεί να χρησιμοποιηθεί δείκτης που δείχνει σε δείκτη
- ✓ Λέγεται Ανάκληση

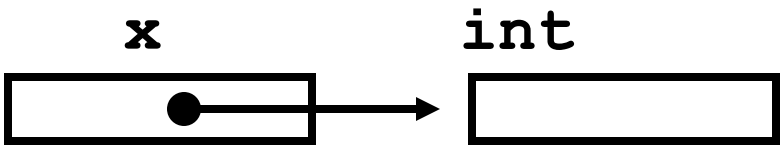
  char **ccptr; // pointer to char pointer

Ανάκληση

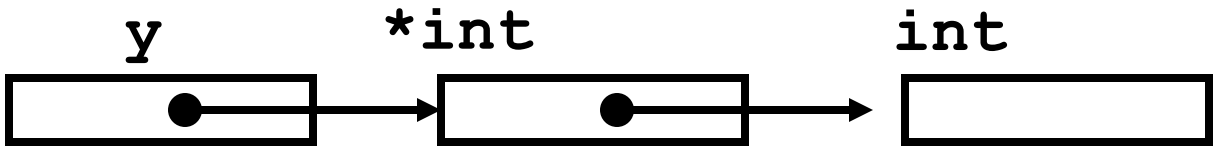

Memory
char *cptr, **ccptr;
cptr = 0xab02;
*cptr = 'A';
ccptr = &cptr;
cout << **ccptr;
0xAB00
0xAB01
'A'
0xAB02
cptr
ccptr
0xAB02
0x00FFAB
0x00FFAB

Ανάκληση

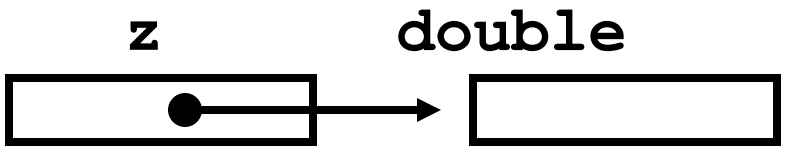
```
int *x;
```




```
int **y;
```




```
double *z;
```



Χρησιμότητα pointer

- ✓ Διαχείριση δεδομένων & μνήμης (δέσμευση μνήμης)
- ✓ Πρόσβαση στα δεδομένα-μέλη κλάσεων και συναρτήσεων
- ✓ Πέρασμα με αναφορά σε συναρτήσεις

Δέσμευση μνήμης

- ✓ Για να διαχειριστούμε σωστά τα δεδομένα και να δεσμεύσουμε μνήμη όπου χρειάζεται χρησιμοποιούμε την εντολή new:

```
unsigned short int * pPointer;
pPointer = new unsigned short int;
```

Ή

```
unsigned short int * pPointer = new unsigned short int;
```

- ✓ Το new ακολουθείται από τον τύπο του αντικειμένου που θέλουμε να δεσμεύσουμε
- ✓ Η διεύθυνση αποδίδεται σε ένα δείκτη
- ✓ Κάθε φορά που αιτούμαστε για μνήμη πρέπει να ελέγχουμε το δείκτη για null (αποτυχία στη δέσμευση)

delete

- ✓ Όταν δεν μας χρειάζεται η δεσμευμένη μνήμη την απελευθερώνουμε με delete:

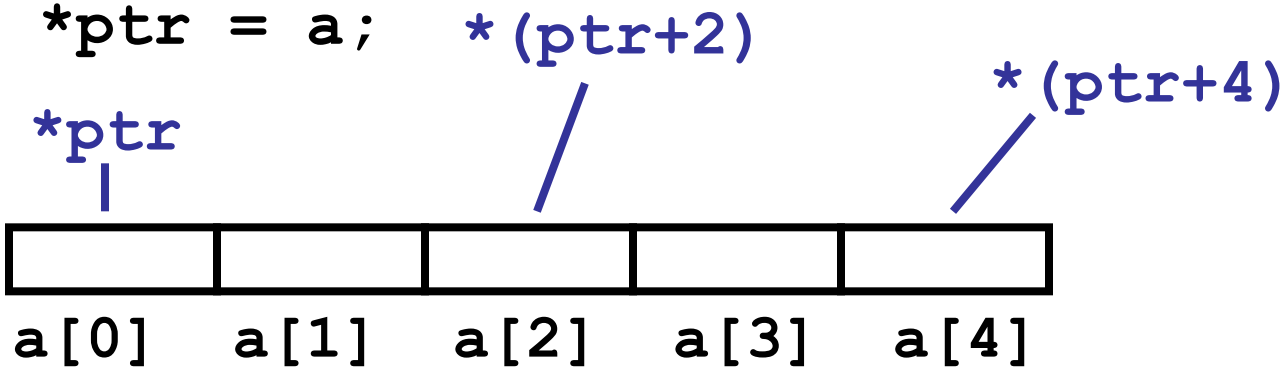
  delete pPointer;

- ✓ Αν ξανακαλέσουμε το delete στον ίδιο δείκτη μπορεί να προκαλέσει crash
- ✓ Για τον ίδιο λόγο καλό είναι μετά το delete να αποδίδουμε στο δείκτη τιμή 0(null)
- ✓ Crash θα προκαλέσει και η προσπάθεια να αποδώσουμε εκ νέου μνήμη σε διαγραμμένο δείκτη.
- ✓ ΠΡΟΣΟΧΗ: το σωστό είναι για κάθε new να υπάρχει ένα delete

Δείκτες (Pointers)

- ✓ Οι δείκτες μπορούν να χρησιμοποιηθούν σε μαθηματικές εκφράσεις
- ✓ Όταν αυξάνουμε ένα δείκτη θα αυξηθεί σύμφωνα με το μέγεθος του αντικειμένου που δείχνει

```
int *ptr = a;
```




```
int a[5];
```

Αναφορές (references)

- ✓ Οι αναφορές έχουν τη δύναμη των δεικτών αλλά με πιο εύκολη σύνταξη
- ✓ Όταν δημιουργούμε μια αναφορά την αρχικοποιούμε με μία άλλη μεταβλητή (στόχο)

```
int &rSomeRef = someInt;
```

- ✓ Η αναφορά λειτουργεί ως εναλλακτικό όνομα για το στόχο και ότι κάνουμε στην αναφορά περνάει στο στόχο

Που ορίζουμε αναφορά

- ✓ Μπορούμε να ορίσουμε αναφορά σε αντικείμενο
- ✓ Δεν μπορούμε να ορίσουμε αναφορά σε κλάση
- ✓ Οι αναφορές πρέπει να αρχικοποιούνται κατά τη δήλωση

```
int hisAge;
int &rAge = hisAge;

CAT boots;
CAT &rCatRef = boots;
```

Πέρασμα με αναφορά

- ✓ Το πέρασμα με αναφορά μπορεί να γίνει με τη χρήση δεικτών ή αναφορών
- ✓ Σε αυτή την περίπτωση αντί να περάσουμε αντίγραφο των δεδομένων στη συνάρτηση, περνάμε τα original δεδομένα.

Overloading τους προκαθορισμένους τελεστές

- ✓ Μπορούμε να κάνουμε overloading του προκαθορισμένους τελεστές δηλώνοντας συναρτήσεις της μορφής

  returnType Operator op (parameters)

- ✓ όπου, op ο τελεστής για overloading

  void operator++ ()

Αρχικοποίηση Αντικειμένων

- ✓ Οι Constructors περιλαμβάνουν δύο τμήματα:

  την αρχικοποίηση και το σώμα.

- ✓ Οι μεταβλητές μπορούν να πάρουν τιμή σε οποιοδήποτε από τα δύο τμήματα:
  - Είτε αρχικοποιώντας τους στο τμήμα αρχικοποίησης
  - Είτε δίνοντας τιμή στο σώμα του

```
CAT():
itsAge(5),              // λίστα αρχικοποίησης
itsWeight(8)
{}                      // σώμα constructor
```

Αρχικοποίηση Αντικειμένων

```
Rectangle::Rectangle():
itsWidth(5), itsLength(10)
{ };

Rectangle::Rectangle (int width, int length):
itsWidth(width), itsLength(length)
{ };
```

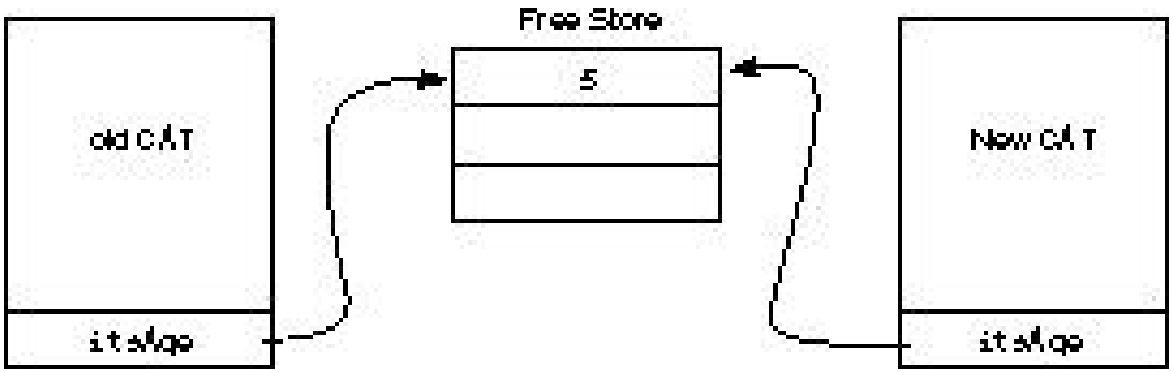
Copy Constructor

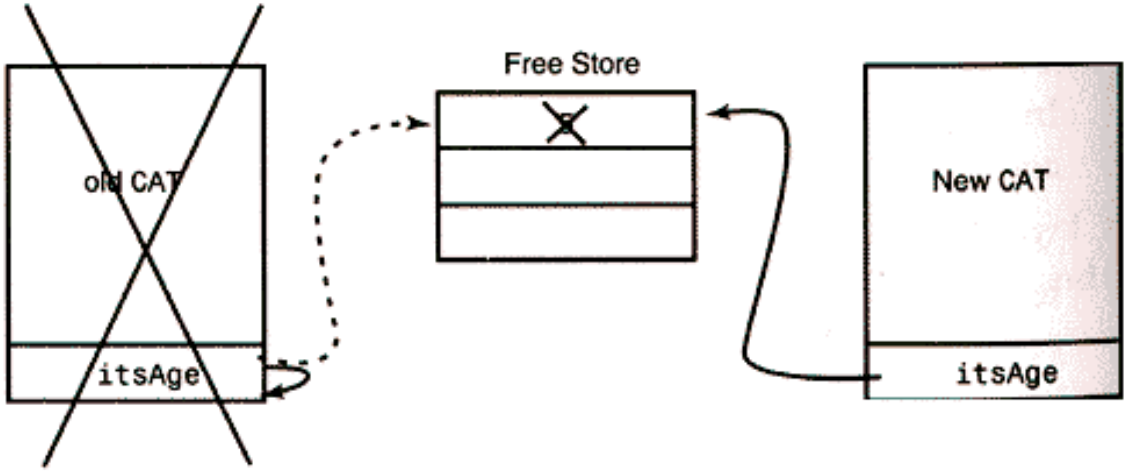
- ✓ Ο compiler διαθέτει τον Copy Constructor για τις περιπτώσεις που χρειαζόμαστε αντίγραφο αντικειμένου, πχ πέρασμα σε συνάρτηση
- ✓ Όλοι οι copy constructors παίρνουν ως παράμετρο μια αναφορά σε αντικείμενο της ίδιας κλάσης.

CAT(const CAT & theCat);

- ✓ Ο default copy constructor αντιγράφει κάθε δεδομένο του αντικειμένου-παραμέτρου στα δεδομένα ενός νέου αντικειμένου.
- ✓ Αυτό μπορεί να είναι πρόβλημα όταν τα δεδομένα είναι δείκτες καθώς θα δείχνουν και οι δύο στην ίδια θέση μνήμης αλλά θα πάψουν να υπάρχουν με την καταστροφή του αντίστοιχου αντικειμένου

Copy Constructor





Copy Constructor

- ✓ Η λύση είναι να δημιουργήσουμε το δικό μας copy constructor που θα δεσμεύσει την απαιτούμενη μνήμη εξ' αρχής πριν αντιγράψει τις τιμές σε νέα μνήμη.

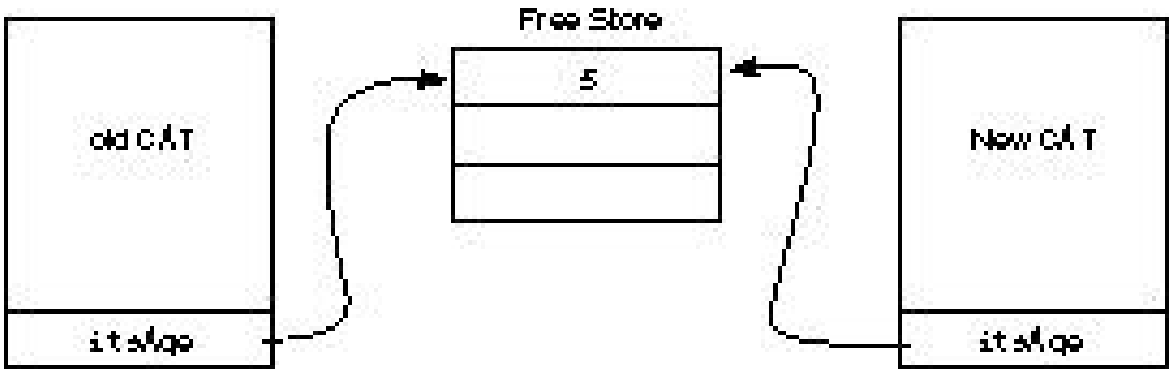
Copy Constructor

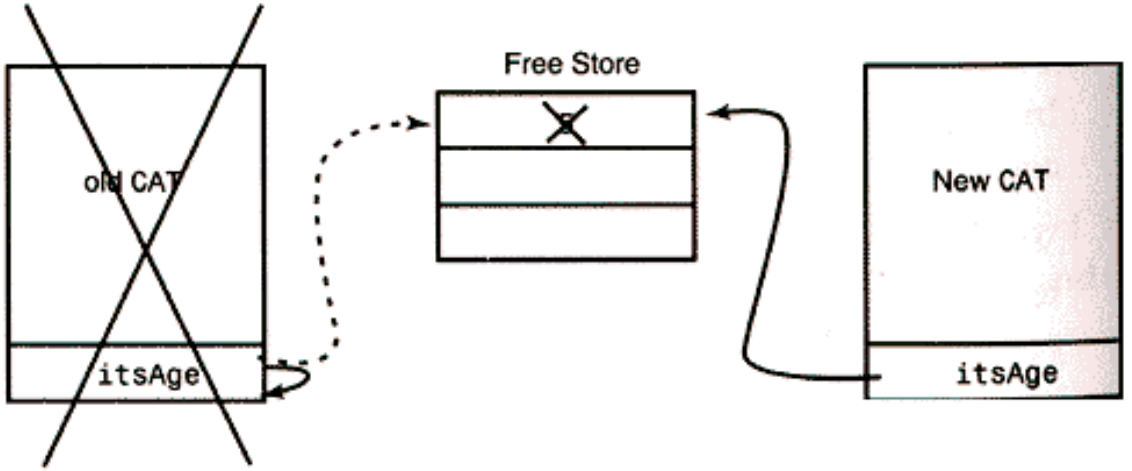
- ✓ Ο compiler διαθέτει τον Copy Constructor για τις περιπτώσεις που χρειαζόμαστε αντίγραφο αντικειμένου, πχ πέρασμα σε συνάρτηση
- ✓ Όλοι οι copy constructors παίρνουν ως παράμετρο μια αναφορά σε αντικείμενο της ίδιας κλάσης.

CAT(const CAT & theCat);

- ✓ Ο default copy constructor αντιγράφει κάθε δεδομένο του αντικειμένου-παραμέτρου στα δεδομένα ενός νέου αντικειμένου.
- ✓ Αυτό μπορεί να είναι πρόβλημα όταν τα δεδομένα είναι δείκτες καθώς θα δείχνουν και οι δύο στην ίδια θέση μνήμης αλλά θα πάψουν να υπάρχουν με την καταστροφή του αντίστοιχου αντικειμένου

Copy Constructor





Copy Constructor

- ✓ Η λύση είναι να δημιουργήσουμε το δικό μας copy constructor που θα δεσμεύσει την απαιτούμενη μνήμη εξ' αρχής πριν αντιγράψει τις τιμές σε νέα μνήμη.

Κλάση Βάση/Παράγωγη

- ✓ Τα διάφορα αντικείμενα μπορούν να έχουν μεταξύ τους σχέση ταξινόμησης π.χ Θηλαστικό - Σκύλος
- ✓ Η C++ δίνει τη δυνατότητα απεικόνισης τέτοιων σχέσεων, ορίζοντας κλάσεις παραγόμενες από άλλες.
- ✓ Για παράδειγμα μπορούμε να ορίσουμε την κλάση Σκύλος ως παράγωγη της κλάσης Θηλαστικό.
- ✓ Σε αυτή την περίπτωση π.χ δεν χρειάζεται να ορίσουμε ότι ο Σκύλος κινείται αν έχουμε ήδη ορίσει ότι το Θηλαστικό κινείται.
- ✓ Η κλάση Θηλαστικό λέγεται Βάση και η Σκύλος Παράγωγη.
- ✓ Μία κλάση Βάση μπορεί να έχει πολλές Παράγωγες.

Παραγόμενες κλάσεις

- ✓ Ας φανταστούμε ότι πρέπει να φτιάξουμε ένα πρόγραμμα που περιγράφει μία φάρμα ζώων.
- ✓ Θα πρέπει να συμπεριλάβουμε κλάσεις για τα διάφορα ζώα.
- ✓ Όταν δηλώνουμε μία κλάση θα πρέπει να υποδεικνύουμε από ποια κλάση παράγεται, γράφοντας μετά το όνομα της κλάσης, τον τρόπο παραγωγής:

```
class Dog : public Mammal
```

- ✓ Η κλάση Βάση **ΠΡΕΠΕΙ** να έχει δηλωθεί νωρίτερα.

Protected δεδομένα

- ✓ Η παραγώμενη κλάση, κληρονομεί από την κλάση βάσης όλα τα δεδομένα και τις συναρτήσεις, **εκτός** από τον τελεστή αντιγραφής, το δομητή και τον αποδομητή.
- ✓ Τα private μέλη δεν είναι διαθέσιμα στις παραγόμενες κλάσεις
- ✓ Τα protected μέλη είναι απολύτως διαθέσιμα στις παραγόμενες κλάσεις και private για το υπόλοιπο πρόγραμμα.
- ✓ Οι συναρτήσεις μέλη έχουν πρόσβαση σε όλα τα δεδομένα και συναρτήσεις της κλάσης τους (ακόμα και τα private) και στα public και protected μέλη των κλάσεων βάσης.

Υπερίσχυση Συναρτήσεων (overriding)

- ✓ Έχουμε υπερίσχυση συναρτήσεων όταν στην παράγωγη κλάση ξαναδημιουργείται μία συνάρτηση της βάσης με τον ίδιο τύπο επιστροφής, όνομα και παραμέτρους και διαφορετική υλοποίηση.
- ✓ Όταν καλείται ένα αντικείμενο της παραγόμενης κλάσης καλείται η νέα συνάρτηση.

Overloading Vs Overriding

- ✓ Αυτοί οι όροι μοιάζουν και κάνουν παρόμοια πράγματα
- ✓ Όταν κάνουμε μία συνάρτηση overloading, δημιουργούμε μία συνάρτηση με το ίδιο όνομα αλλά διαφορετικές παραμέτρους.
- ✓ Το overriding δημιουργεί στην παραγόμενη κλάση μία συνάρτηση με τα ίδια όνομα, παραμέτρους και τύπο επιστροφής.

Απόκρυψη συναρτήσεων

- ✓ Συμβαίνει απόκρυψη συνάρτησης όταν παραλείψουμε οποιοδήποτε τμήμα της κεφαλίδας της, ακόμα και αν είναι μόνο η λέξη const
- ✓ Αν έχουμε overriding σε μία συνάρτηση της βάσης, μπορούμε και πάλι να την καλέσουμε αν γράψουμε το πλήρες όνομα:

Mammal::Move()

Virtual Συναρτήσεις

- ✓ Μία Virtual συνάρτηση είναι συνάρτηση της βασικής κλάσης που μπορεί να υπερκαλυφθεί από συνάρτηση της παραγόμενης κλάσης
- ✓ Σύνταξη

  virtual ret_type FunctionName(args)

Abstract Classes

- ✓ Μια κλάση που έχει τουλάχιστο μία καθαρά virtual συνάρτηση λέγεται αφαίρετική κλάση (abstract class).

Κανόνες πρόσβασης Κληρονομικότητας

- ✓ Τα **private** μέλη δεν κληρονομούνται
- ✓ Τα **protected** μέλη κληρονομούνται, αλλά δεν είναι ορατά εκτός κλάσης
- ✓ Η C++ έχει 3 επίπεδα ελέγχου πρόσβασης
- ✓ Σύνταξη: **class B *: είδος πρόσβασης* A {...};**
- ✓ Τα τρία επίπεδα είναι:

  public: **public** παραμένει **public**, **protected** παραμένει **protected**

  protected: **public** γίνεται **protected**, **protected** μένει **protected**

  private: **public** and **protected** γίνονται **private**

Ανάγκη για Πολλαπλή Κληρονομικότητα

- ✓ Ας υποθέσουμε ότι με τη βοήθεια της Ιεραρχίας έχουμε χωρίσει την κλάση των ζώων σε πουλιά και θηλαστικά.
- ✓ Η κλάση των πουλιών, Bird, εμπεριέχει τη συνάρτηση Fly(), ενώ στα θηλαστικά ανήκει η κλάση Horse που περιέχει τις συναρτήσεις Whinny() και Gallop().
- ✓ Έστω ότι χρειαζόμαστε την κλάση Pegasus που χρειάζεται τις συναρτήσεις Fly(), Whinny(), και Gallop(). Ποια πρέπει να είναι η βάση μας;

πολλαπλή κληρονομικότητα

- ✓ Είναι δυνατό να παράγουμε μία κλάση από δύο βάσεις:

```
class Pegasus : public Horse, public Bird
```

- ✓ Αυτό ονομάζεται πολλαπλή κληρονομικότητα.
- ✓ Τότε η παραγόμενη κληρονομεί και από τις δύο βάσεις

Διπλή συνάρτηση

- ✓ Στην περίπτωση που μία συνάρτηση υπάρχει σε δύο κλάσεις βάσης μιας παραγόμενης κλάσης και κληθεί από την παραγόμενη κλάση, θα προκύψει σφάλμα.
- ✓ Το πρόβλημα μπορεί να ξεπεραστεί με δύο τρόπους:
  - Είτε ορίζοντας ακριβώς ποια συνάρτηση καλούμε

    COLOR currentColor = pPeg->Horse::GetColor();
  - Είτε κάνοντας override τη συνάρτηση στην παραγόμενη κλάση

Virtual Κληρονομικότητα

- ✓ Η χρήση μιας virtual κλάσης βάσης μας επιτρέπει τη χρήση κοινών συναρτήσεων ορισμένων ως virtual στη βάση, χωρίς να δημιουργεί πολλαπλά αντίγραφα.
- ✓ Κανονικά, ο δομητής μιας κλάσης, αρχικοποιεί μόνο τις μεταβλητές της κλάσης του και τη βάση του.
- ✓ Οι virtual κλάσεις βάσης αποτελούν εξαίρεση καθώς αρχικοποιούνται και από την τελευταία στη σειρά παραγόμενη κλάση.

Δήλωση κλάσεων για virtual κληρονομικότητα

- ✓ Για να διασφαλίσουμε ότι οι παραγόμενες κλάσεις διατηρούν ένα μόνο αντίγραφο κοινών συναρτήσεων των κλάσεων βάσης, δηλώνουμε τις κλάσεις βάσης ως παραγόμενες από μια virtual βάση.

Πολλαπλή Κληρονομικότητα

- ✓ Χρησιμοποιήστε Πολλαπλή Κληρονομικότητα όταν μία κλάση χρειάζεται συναρτήσεις και στοιχεία από περισσότερες της μιας κλάσης.
- ✓ Υπάρχουν compilers που δεν υποστηρίζουν την πολλαπλή κληρονομικότητα.
- ✓ Μη χρησιμοποιείται πολλαπλή κληρονομικότητα όταν δε χρειάζεται ή όταν σας καλύπτει η απλή κληρονομικότητα γιατί αυξάνει την πολυπλοκότητα του προγράμματος
- ✓ Αρχικοποιήστε τη virtual κλάση βάσης από την τελευταία στην ιεραρχία παραγόμενη κλάση.

Mixins classes

- ✓ Η Mixin class προσθέτει λειτουργικότητα ενώ δεν προσθέτει πολλά δεδομένα.
- ✓ Οι mixin classes προστίθενται σε μία παραγόμενη κλάση με public κληρονομικότητα.
- ✓ Αποτελούν συμβιβασμό μεταξύ απλής και πολλαπλής κληρονομικότητας.
- ✓ Η διαφορά τους από τις άλλες κλάσεις είναι ότι περιέχουν ελάχιστα ή καθόλου δεδομένα.

Pure Virtual συναρτήσεις

- ✓ Μια virtual συνάρτηση γίνεται pure virtual όταν αρχικοποιείται στο 0.

```
virtual void Draw() = 0;
```

- ✓ Κάθε κλάση με μία ή περισσότερες Pure Virtual συναρτήσεις αποτελεί Αφηρημένη κλάση
- ✓ Δεν μπορούμε να δημιουργήσουμε αντικείμενο αφηρημένης κλάσης
- ✓ Είναι υποχρεωτικό να ορίσουμε τις Pure Virtual συναρτήσεις σε κάθε παραγόμενη κλάση

Ένα ολοκληρωμένο παράδειγμα

```
#include <iostream>
#include <math.h>
#define PI 3.14
using namespace std;

/* κλάση σχήμα */
class Shape {
public:
        virtual ~Shape() {}
        virtual double perimetros() = 0;
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* κλάση για τον ορισμό χρώματος */
class Colorfull {
private:
        int r;
        int g;
        int b;
public:
        virtual ~Colorfull() {}
        void setRed(const int red) { r = red; }
        void setGreen(const int green) { g = green; }
        void setBlue(const int blue) { b = blue; }
        int getRed() const { return r; }
        int getGreen() const { return g; }
        int getBlue() const { return b; }
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* κλάση για τον ορισμό σημείου */
class Point {
private:
        double x;
        double y;
public:
        Point(const double _x, const double _y) : x(_x), y(_y) {}
        Point() : x(0), y(0) {}
        //  Point(const Point &p) : x(p.x), y(p.y) {}
        double distance(const Point& p) {
                double diffX = x - p.x;
                double diffY = y - p.y;
                return sqrt(diffX * diffX + diffY * diffY);
}
 ~Point() {}
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* κλάση Τρίγωνο */
class Triangle : public Shape {
private:
//Οι συντεταγμένες
        Point p1;
        Point p2;
        Point p3;
public:
Triangle(const Point& _p1, const Point& _p2, const Point& _p3)   :
p1(_p1), p2(_p2), p3(_p3) {}
~Triangle() {}
double perimetros(){  return p1.distance(p2) + p1.distance(p3) +
p2.distance(p3);    }
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* κλάση Τετράγωνο */
class Square : public Shape {
private:
        Point pos;
        double edgeSize;
public:
        Square(const Point& p, const double edge) : pos(p),
                edgeSize(edge) {}
        ~Square() {}
        double perimetros() {  return 4 * edgeSize;    }
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* κλάση Κύκλος */
class Circle : public Shape {
private:
        Point center;
        double aktina;
public:
        Circle(const Point& c, const double radius) : center(c),
                aktina(radius) {}
        ~Circle() {}
        double perimetros(){  return 2 * PI * aktina;}
};
```

Ένα ολοκληρωμένο παράδειγμα

```
/* Ένα τρίγωνο σχεδιασμένο με χρώμα */
class ColorTriangle : public Triangle, public Colorfull {
public:
        ColorTriangle(const Point& _p1, const Point& _p2, const
        Point& _p3) : Triangle(_p1, _p2, _p3), Colorfull() {}
};

/* Ένα τετράγωνο σχεδιασμένο με χρώμα */
class ColorSquare : public Square, public Colorfull {
public:
        ColorSquare(const Point& p, const double edge) : Square(p,
        edge), Colorfull() {}
};
```

Ένα ολοκληρωμένο παράδειγμα

```
int main()
{
Point circle_p(10,10);
Circle c1(circle_p, 2);
cout << c1.perimetros() << endl;

Point square_p(20,20);
Square sq1(square_p, 6);
cout << sq1.perimetros() << endl;

Point square_p2(30,30);
ColorSquare sq2(square_p2, 10);
sq2.setBlue(5);
sq2.setGreen(9);
sq2.setRed(10);
cout << sq2.perimetros() << endl;}
```