Η ΓΛΩΣΣΑ C++

Μάθημα 12:

Πολλαπλή Κληρονομικότητα

Δημήτρης Ψούνης

www.psounis.gr

Περιεχόμενα Μαθήματος

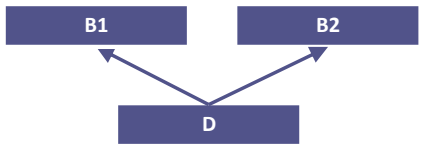
Α. Θεωρία

1. Πολλαπλή Κληρονομικότητα

1. Ορισμός Πολλαπλής Κληρονομικότητας

- Ορίζουμε ότι μία κλάση D κληρονομεί από δύο κλάσεις B1 και B2 με τις δηλώσεις:

```
class B1 {
  …
};

class B2 {
  …
};

class D: public B1, public B2 {
 …
};
```



- Με παρόμοιο τρόπο μπορούμε να ορίσουμε μία κλάση να κληρονομεί από περισσότερες κλάσεις
- Όταν μία κλάση κληρονομεί δύο ή περισσότερες κλάσεις, λέμε ότι έχουμε **πολλαπλή κληρονομικότητα**.

Α. Θεωρία

1. Πολλαπλή Κληρον/τα

2. Παράδειγμα

- Το ακόλουθο παράδειγμα, υλοποιεί πολλαπλή κληρονομικότητα και μας γυρνάει πίσω στην εποχή του Μάρκου Αυρήλιου (121-180 μΧ):

```
class king {
   public:
      void set_realm(string in_realm);
      void rule();
   private:
      string realm;
};

void king::set_realm(string in_realm)
{
  realm = in_realm;
}

void king::rule()
{
   cout<<"Now, I rule"<<endl;
}
```

```
class philosopher {
   public:
      void set_philosophy(string in_philosophy);
      void think();
   private:
      string philosophy;
};

void philosopher::set_philosophy(string in_philosophy)
{
   philosophy = in_philosophy;
}

void philosopher::think()
{
   cout<<"Now, I think"<<endl;
}
```

Α. Θεωρία

1. Πολλαπλή Κληρονομικότητα

2. Παράδειγμα

«Θα υπάρχουν βάτοι στο δρόμο. Απέφυγε τους. Μην σκέφτεσαι "γιατί υπάρχουν τέτοια πράγματα στον κόσμο;"»

Μάρκος Αυρήλιος(121-180μΧ)

```
class phil_king: public king, public philosopher {
   public:
      void routine();
};

void phil_king::routine()
{
   think();
   rule();
   think();
}
```

```
int main()
{
   phil_king marcus_aurelius;

   marcus_aurelius.set_realm("Roman Empire");
   marcus_aurelius.set_philosophy("stoic");

   marcus_aurelius.routine();
}
```

Ολοκληρωμένο το πρόγραμμα είναι το: «cpp12.multiple_inheritance.cpp»

Α. Θεωρία

2. Κατασκευαστές - Καταστροφείς

1. Σειρά Εκτέλεσης Κατασκευαστών - Καταστροφέων

- Η σειρά εκτέλεσης των κατασκευαστών των βασικών κλάσεων είναι:
  - Από αριστερά προς τα δεξιά, όπως τις γράψαμε κατά τη δήλωση της παραγόμενης κλάσης.
- Η σειρά εκτέλεσης των καταστροφέων είναι
  - Αντίστροφη, δηλαδή από δεξιά προς τα αριστερά, όπως τις γράψαμε κατά τη δήλωση της παραγόμενης κλάσης.

- Υπενθύμιση:
  - Πρώτα τρέχει ο κατασκευαστής της βασικής κλάσης και έπειτα της παραγόμενης κλάσης.

Α. Θεωρία

2. Κατασκευαστές - Καταστροφείς

2. Λίστες Ορισμάτων και Πολλαπλή Κληρονομικότητα

- Η αρχικοποίηση των κατασκευαστών γίνεται με λίστες ορισμάτων από αριστερά προς τα δεξιά σύμφωνα με τους κατασκευαστές των βασικών κλάσεων.
- Π.χ.:

```
D::D(args for B1, args for B2, args for D):
   B1(args for B1), B2(args for B2)
{
   …
}
```


Α. Θεωρία

2. Κατασκευαστές - Καταστροφείς

3. Παράδειγμα

```
class king {
   public:
      king(string in_realm);
      ~king();
      …
};

class philosopher {
   public:
      philosopher(string in_philosophy);
      ~philosopher();
      …
};

class phil_king: public king, public philosopher {
   public:
      phil_king(string in_realm,
                string in_philosophy);
      ~phil_king();
};
```

βλ. «cpp12.mi_constructors.cpp»

```
king::king(string in_realm)
{
   realm = in_realm;
   cout<<"Constructing King"<<endl;
}
king::~king()
{
   cout<<"Destructing King"<<endl;
}
philosopher::philosopher(string in_philosophy)
{
   philosophy = in_philosophy;
   cout<<"Constructing Philosopher"<<endl;
}
philosopher::~philosopher()
{
   cout<<"Destructing Philosopher"<<endl;
}
phil_king::phil_king(string in_realm, string in_philosophy):
   king(in_realm), philosopher(in_philosophy)
{
   cout<<"Constructing Philosopher King"<<endl;
}
phil_king::~phil_king() {
   cout<<"Destroying Philosopher King"<<endl;
}
```

Α. Θεωρία
2. Κατασκευαστές - Καταστροφείς
3. Παράδειγμα

Και η εκτύπωση που θα δώσει η παρακάτω main:

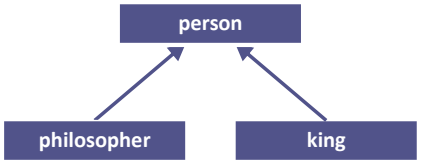
```
int main()
{
   phil_king marcus_aurelius("Roman Empire", "stoic");

   marcus_aurelius.routine();
}
```

είναι:

```
Constructing King
Constructing Philosopher
Constructing Philosopher King
Now, I think
Now, I rule
Now, I think
Destroying Philosopher King
Destructing Philosopher
Destructing King
```


Α. Θεωρία
3. Το πρόβλημα του διαμαντιού
1. Επέκταση της οντολογίας του φιλοσόφου - βασιλιά

- Επιθυμόντας να προσθέσουμε το όνομα του φιλοσόφου-βασιλιά μέσα στην κλάση
  - αντιλαμβανόμαστε ότι αυτό είναι ένα πιο γενικό χαρακτηριστικό
  - και αποφασίζουμε να κατασκευάσουμε μια κλάση άτομο που θα κληρονομείται τόσο από το φιλόσοφο, όσο και από το βασιλιά
    - και θα περιέχει το όνομα ως μέλος του.



Α. Θεωρία
3. Το πρόβλημα του διαμαντιού
1. Επέκταση της οντολογίας του φιλοσόφου - βασιλιά

- Τροποποιούμε το πρόγραμμα μας με τη νέα κλάση:

```
class person {
   public:
      person(string in_name);
   protected:
      string name;
};
```

```
person::person(string in_name)
{
   name = in_name;
}
```

- την οποία κληρονομούν κατάλληλα οι δύο κλάσεις (βασιλιάς και φιλόσοφος)
- και προσθέτουμε στην κλάση phil_king υπερφόρτωση του τελεστή <<

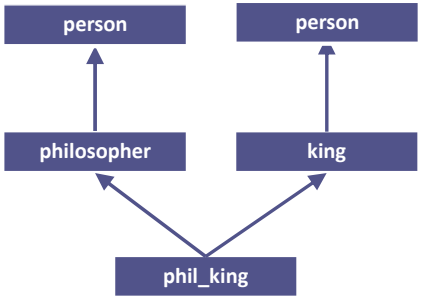
```
ostream &operator<<(ostream &left, phil_king right)
{
   left<<right.name<<endl;
   left<<"Ruling "<<right.realm<<endl;
   left<<"using "<<right.philosophy<<" philosophy"<<endl;
}
```

- Ο μεταγλωττιστής διαμαρτύρεται…

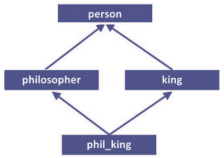
βλ. «cpp12.diamond_problem.cpp»

Α. Θεωρία
3. Το πρόβλημα του διαμαντιού
1. Επέκταση της οντολογίας του φιλοσόφου - βασιλιά

- Το πρόβλημα συνίσταται στην οντολογία μας:
  - Στην πραγματικότητα έχουμε υλοποιήσει το σχήμα:



- Δηλαδή έχουμε δύο αντικείμενα person στην κλάση μας.
- Οπότε ο μεταγλωττιστής δεν ξέρει ποιο από τα δύο person::name να χρησιμοποιήσει.
- Το πρόβλημα αυτό, αναφέρεται ως το «πρόβλημα του διαμαντιού»
  - Μία κλάση κληρονομεί, από δυο κλάσεις που κληρονομούν από την ίδια κλάση



Α. Θεωρία

3. Το πρόβλημα του διαμαντιού

2. Πρώτη Λύση: Χρήση του τελεστή εμβέλειας

- Μία λύση είναι να καθορίσουμε ρητά σε ποια κλάση αναφερόμαστε χρησιμοποιώντας τον τελεστή εμβέλειας.
- Στο παράδειγμα αλλάζουμε την προβληματική αναφορά ως εξής:

```
ostream &operator<<(ostream &left, phil_king right)
{
   left<<right.philosopher::name<<endl;<<endl;
   left<<"Ruling "<<right.realm<<endl;
   left<<"using "<<right.philosophy<<" philosophy"<<endl;
}
```

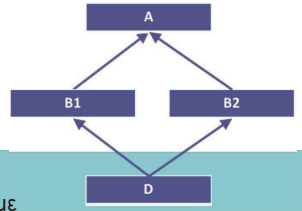
- Σημείωση:
  - Ο τελεστής επίλυσης εμβέλειας μπορεί να χρησιμοποιείται οποτεδήποτε θέλουμε να έχουμε πρόσβαση σε κάποιο μέλος ή μέθοδο όταν υπάρχουν συγκρούσεις.

βλ. «cpp12.use_of_resolution_scope_operator.cpp»

Α. Θεωρία

3. Το πρόβλημα του διαμαντιού

3. Δεύτερη Λύση: Εικονική Κληρονομικότητα



- Η δεύτερη λύση είναι οντολογικά ορθότερη:

Εικονική Κληρονομικότητα:
- Για να αποφύγουμε το πρόβλημα του διαμαντιού ορίζουμε
- Τις κλάσεις που κληρονομούν (B1, B2) την κοινή κλάση (A), να την κληρονομούν με virtual τρόπο
  - Τότε στην κοινή υποκλάση τους (D), θα περιέχεται μόνο ένα αντικείμενο της κοινής υπερκλάσης (A)

- Συνεπώς οι δηλώσεις των κλάσεων θα είναι:

```
class person
{ … };
class philosopher: virtual public person
{ … };
class king: virtual public person
{ … };
class phil_king: public king, public philosopher
{ … };
```


Α. Θεωρία

3. Το πρόβλημα του διαμαντιού

3. Δεύτερη Λύση: Εικονική Κληρονομικότητα

- Ενώ τον κατασκευαστή της κοινής υπερκλάσης θα τον καλεί μόνο το αντικείμενο της κοινής υποκλάσης
  - (και δεν θα ασχολούνται μαζί του, οι ενδιάμεσοι)
  - Έτσι οι κατασκευαστές θα είναι:

```
person::person()
{}

person::person(string in_name)
{
   name = in_name;
}
```

```
king::king(string in_realm)
{
   realm = in_realm;
}
philosopher::philosopher(string in_philosophy)
{
   philosophy = in_philosophy;
}
```

```
phil_king::phil_king(string in_name, string in_realm, string in_philosophy):
   person(in_name), king(in_realm), philosopher(in_philosophy)
{ }
```

βλ. «cpp12.virtual_class.cpp»

Α. Θεωρία

3. Το πρόβλημα του διαμαντιού

3. Δεύτερη Λύση: Εικονική Κληρονομικότητα

- Προσοχή!
  - Η λέξη-κλειδί virtual που χρησιμοποιείται για την εικονική κληρονομικότητα, δεν υπήρξε και η καλύτερη επιλογή από τους κατασκευαστές της C++
  - Αρχικά είχαν οριστεί οι εικονικές μέθοδοι (βλέπε προηγούμενο μάθημα) και στη συνέχεια:
    - προτιμήθηκε να διατηρηθεί η λέξη-κλειδί virtual και για αυτήν την περίπτωση
    - παρόλο που το νόημα των δύο ενεργειών δεν ομοιάζει.
  - Πολλές συζητήσεις έχουν γίνει γι΄αυτό το θέμα
    - Το συμπέρασμα είναι ότι θα ήταν ίσως προτιμότερο να έχει ένα άλλο όνομα, αλλά προτιμήθηκε αυτό για να μην αυξάνονται υπερβολικά οι χρησιμοποιούμενες λέξεις κλειδιά.

Συμπέρασμα:
- Ο δύο έννοιες:
  - εικονικές μέθοδοι
  - εικονική κληρονομικότητα
- δεν θα πρέπει να συγχέονται, είναι άσχετες η μία με την άλλη παρόλο που χρησιμοποιούν την ίδια λέξη-κλειδί.

Α. Θεωρία
4. Παρατηρήσεις για την πολλαπλή κληρονομικότητα

- Είδαμε το πρόβλημα (και τη λύση) για το πρόβλημα του διαμαντιού.
- Γενικά έχει παρατηρηθεί ότι η πολλαπλή κληρονομικότητα, έχει αρκετά προβλήματα:
  - Η αποσφαλμάτωση γίνεται δυσκολότερη
  - Είναι πιο δύσκολο να κατασκευαστούν αξιόπιστες ιεραρχίες κλάσεων
  - Αρκετοί προγραμματιστές ισχυρίζονται ότι οτιδήποτε μπορεί να γίνει με την πολλαπλή κληρονομικότητα, μπορεί να γίνει και με απλή κληρονομικότητα.
- Σημειωτέον ότι οι επόμενες γενιές γλωσσών (Java και C#) απέβαλλαν την πολλαπλή κληρονομικότητα από τη λειτουργικότητά τους.

Συμπέρασμα:
- Η πολλαπλή κληρονομικότητα, ιδίως για πολύ μεγάλα προγράμματα, θα πρέπει να αποφεύγεται.

Α. Θεωρία
5. Άλλοι Τύποι Κληρονομικότητας

- Εκτός από τη δημόσια κληρονομικότητα, υπάρχει και η ιδιωτική κληρονομικότητα
- Συνοψίζουμε την πρόσβαση που ορίζεται στην παραγόμενη κλάση ανάλογα με το είδος κληρονομικότητας:

Είδος Κληρονομικότητας

Επίπεδο πρόσβασης χαρακτηριστικών βασικής κλάσης	public	protected	private
base::public	public	protected	private
base::protected	protected	protected	private
base::private	N/A	N/A	N/A

Επίπεδο Πρόσβασης στην Παραγόμενη Κλάση

Παρατήρηση:
- Οι άλλοι τύποι κληρονομικότητας (private-protected) είναι χρήσιμοι μόνο σε εξειδικευμένες περιπτώσεις και ξεφεύγουν από τα όρια του μαθήματος.

Β. Ασκήσεις
Άσκηση 1.1: Ιδιοκτήτης Κινηματογράφου

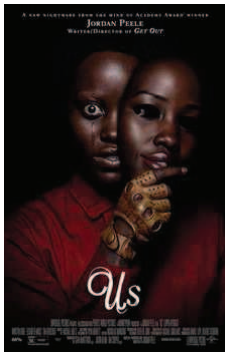
Ένας ιδιοκτήτης κινηματογράφου επαρχιακής πόλης κατασκευάζει μια οντολογία ταινιών για να επιλέγει αλγοριθμικά την ταινία που θα προβάλλει ο κινηματογράφος του την επόμενη εβδομάδα

- Ορίστε μία κλάση movie
  - Αποθηκεύει το όνομα της ταινίας, το σκορ της στο imdb (δεκαδικός) και πριν πόσες εβδομάδες έκανε πριεμιέρα.
  - Να έχει κατασκευαστή
  - Να έχει μία μέθοδο: score
    - Επιστρέφει το πηλίκο του σκορ του στο imdb προς τον όρο (0,5 * w) όπου w το πλήθος των εβδομάδων που έκανε πρεμιέρα.
- Ορίστε μία κλάση thriller που επεκτείνει την κλάση film
  - Αποθηκεύει πρόσθετα τον δεκαδικό fear_factor (0-10) που αναδεικνύει το πόσο τρομακτική είναι η ταινία.
  - Έχει κατασκευαστή
  - Να επαναορίζει τη μέθοδο score:
    - Να επιστρέφει το γινόμενο του fear_factor με το score της ταινίας.

Β. Ασκήσεις
Άσκηση 1.1: Ιδιοκτήτης Κινηματογράφου

Κατασκευάστε κατάλληλη συνάρτηση main που να τυπώνει το σκορ για τις ακόλουθες ταινίες:

IT, Chapter 2
10 weeks
IMDB Score: 6,8
Fear-Factor: 8,0



Us
30 weeks
IMDB Score: 6,9
Fear-Factor: 7,0

Silence of the Lambs
1540 weeks
IMDB Score: 8,6
Fear-Factor: 9,0

Ποια ταινία θα επιλέξει ο αιθουσάρχης;

Β. Ασκήσεις

Άσκηση 1.2: Ιδιοκτήτης Κινηματογράφου

Ο αιθουσάρχης επεκτείνει το πρόγραμμα του με την κλάση comedy

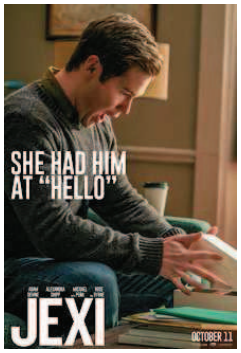
- Ορίστε μία κλάση thriller που επεκτείνει την κλάση film
  - Αποθηκεύει πρόσθετα τον δεκαδικό fun_factor (0-10) που αναδεικνύει το πόσο τρομακτική είναι η ταινία.
  - Έχει κατασκευαστή
  - Να επαναορίζει τη μέθοδο score:
    - Να επιστρέφει το γινόμενο του fun_factor με το score της ταινίας.

- Ορίστε μία κλάση comedy_thriller που έχει τα χαρακτηριστικά τόσο κωμωδίας, όσο και θρίλερ
  - Έχει κατασκευαστή
  - Να επαναορίζει τη μέθοδο score:
    - Να επιστρέφει το γινόμενο του ημιαθροίσματος του fun_factor με το fear_factor, με το score της ταινίας.

Β. Ασκήσεις

Άσκηση 1.2: Ιδιοκτήτης Κινηματογράφου

Επεκτείνετε τη συνάρτηση main που να τυπώνει το σκορ για τις ακόλουθες ταινίες:



Jexi
4 weeks
IMDB Score: 6,6
Fun-Factor: 3,0

Addams Family
5 weeks
IMDB Score: 5,9
Fear-Factor: 3,0
Fun-Factor: 4,0

Shaun of the Dead
825 weeks
IMDB Score: 7,9
Fear-Factor: 4,0
Fun-Factor: 9,0

Ποια ταινία θα επιλέξει ο αιθουσάρχης;

Β. Ασκήσεις

Άσκηση 2.1: Προσομοίωση Καφετέριας

Ορίστε μία κλάση person

- Μέλη: Όνομα, Μισθός

Ορίστε μία κλάση waiter

- Κληρονομεί την person
- Έχει μία μέθοδο serve (παίρνει όρισμα πόσους πελάτες εξυπηρετεί και το barista που μεταφέρει την παραγγελία)
- Κρατάει σε μία μεταβλητή τους πελάτες που έχει εξυπηρετήσει (να αρχικοποιείται μέσω του κατασκευαστή σε 0)

Ορίστε μία κλάση barista

- Κληρονομεί την person
- Έχει μία μέθοδο prepare (καλείται από τη serve του waiter)

Ορίστε μία κλάση owner

- Κληρονομεί την waiter και την barista (κάνει και τις δύο δουλειές)

Β. Ασκήσεις

Άσκηση 2.2: Προσομοίωση Καφετέριας

Στην main:

- Ορίζονται ένας ιδιοκτήτης, δύο σερβιτόροι και ένας μπαρίστα
- Κατασκευάζονται δύο πίνακες δεικτών:
  - Ο πρώτος να αποθηκεύει τους δυνατούς σερβιτόρους
  - Ο δεύτερος να αποθηκεύει τους δυνατούς μπαρίστα

- Επαναληπτικά για 100 φορές:
  - Έρχεται μία παρέα (τυχαίο πλήθος ατόμων από 1 έως 5)
  - Επιλέγεται τυχαία ένας από τους σερβιτόρους
    - ο οποίος στη συνέχεια επιλέγει τυχαία έναν από τους μπαρίστα
  - Γίνεται η εξυπηρέτηση

- Στο τέλος οι σερβιτόροι και ο ιδιοκτήτης να τυπώνουν πόσα άτομα εξυπηρέτησε ο καθένας.