

# Πολυμορφισμός και ιδεατές συναρτήσεις

#15

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Ιωαννίνων (Άρτα)

Γκόγκος Χρήστος

# Ανάγκη ύπαρξης ιδεατών συναρτήσεων

- Στο παράδειγμα με τους σπουδαστές (lect14/sample5.cpp) για κάθε τύπο σπουδαστή (Student, Grad και Undergrad) εκτυπώνεται διαφορετική έκθεση προόδου.
  - ```
Student s;  
s.GradeReport();  
Grad g;  
g.GradeReport();  
Undergrad u;  
u.GradeReport();
```
- Έστω ότι έχουμε μια λίστα με 3000 σπουδαστές και προσπαθούμε με τον ακόλουθο κώδικα να εμφανίσουμε τις εκθέσεις προόδου για όλους τους σπουδαστές:
  - ```
Student list[3000];  
for(i=0; i<3000; i++)  
    list[i].GradeReport();
```
- Υπάρχουν δύο προβλήματα με τον παραπάνω κώδικα:
  - Η λίστα είναι ένας ομογενής πίνακας, πρέπει να χρησιμοποιήσουμε τον πλέον κοινό τύπο (το βασικό τύπο). Ο βασικός τύπος δεν έχει όλες τις πληροφορίες για τις παραγόμενες κλάσεις Grad και Undergrad προκειμένου να παράγει την αναφορά που χρειάζεται.
  - Η συνάρτηση GradeReport που καλείται είναι η έκδοση που υπάρχει στην κλάση Student (η γενική έκθεση προόδου), και όχι η εξειδικευμένη συνάρτηση που υπάρχει σε κάθε παραγόμενη κλάση.

# Ανάγκη ύπαρξης ιδεατών συναρτήσεων

- Μια πιθανή λύση είναι να χρησιμοποιηθούν δείκτες έτσι ώστε να δημιουργηθεί μια ετερογενής λίστα.
  - Ένας δείκτης μπορεί να δείχνει προς έναν τύπο δεδομένων
  - Ένας δείκτης προς τη βασική κλάση μπορεί να δείχνει προς αντικείμενα που παράγονται από αυτή τη βασική κλάση καθώς ένα αντικείμενο παραγόμενης κλάσης είναι (IS-A) αντικείμενο και της βασικής κλάσης.
    - `Graduate gs; Undergraduate ugs; Student *ps = &gs; ps = &ugs;`
  - Ομοίως μια μεταβλητή αναφορά του βασικού τύπου μπορεί να αναφέρεται προς κάποιο αντικείμενο το οποίο είναι παραγόμενο από τη βασική κλάση.
    - `Graduate gs; Student &rs = gs; Undergraduate ugs; Student &rs2 = ugs;`
- Συνεπώς, μια λίστα με 3000 σπουδαστές μπορεί να δημιουργηθεί ως εξής:
  - `Student *list[3000]; // πίνακας με δείκτες προς σπουδαστές`  
`list[0] = &gs; // χρησιμοποιώντας ένα αντικείμενο Graduate που έχει ήδη οριστεί`  
`list[1] = &us; // χρησιμοποιώντας ένα αντικείμενο Undergraduate που έχει ήδη οριστεί`  
`list[2] = new Graduate; // δυναμικά δεσμεύοντας μνήμη για ένα αντικείμενο Graduate`

# Ανάγκη ύπαρξης ιδεατών συναρτήσεων

- Διάσχιση της λίστας όλων των σπουδαστών και εκτύπωση της έκθεσης προόδου για κάθε σπουδαστή.
- Μια πιθανή λύση:
  - `Student *list[3000];`
  - `...`
  - `for (int i=0;i<size;i++)`  
    `list[i]->GradeReport();`
- Καθώς το κάθε στοιχείο της λίστας δείχνει προς διαφορετικό αντικείμενο, υπάρχει το ερώτημα σχετικά με το ποια έκδοση της `GradeReport()` καλείται στην `list[i]->GradeReport();`

# Ανάγκη ύπαρξης ιδεατών συναρτήσεων

```
Student *list[3000];  
for (int i=0;i<size;i++)  
    list[i]->GradeReport(); // Ποια έκδοση της GradeReport() καλείται;
```

- Η απόφαση σχετικά με τη συνάρτηση που θα καλείται λαμβάνεται κατά τη μεταγλώττιση. Καθώς τα αντικείμενα `list[i]` είναι τύπου `Student*`, θα κληθεί η `GradeReport()` της κλάσης `Student`.
- Αυτό συμβαίνει διότι ο μεταγλωττιστής θα πρέπει να λάβει απόφαση για το ποια συνάρτηση θα καλεί πριν το πρόγραμμα εκτελεστεί (στατική δέσμευση = static binding).
- Για να λυθεί το πρόβλημα χρειαζόμαστε έναν τρόπο με τον οποίο θα γίνει dynamic binding (δυναμική δέσμευση) κατά την εκτέλεση, δηλαδή η συνάρτηση να γίνεται bind σε διαφορετικό κώδικα ανάλογα με την περίπτωση.
  - Οι ιδεατές συναρτήσεις (virtual functions) είναι ο μηχανισμός με τον οποίο μπορεί να επιτευχθεί ακριβώς αυτό στη C++.

# Ιδεατές συναρτήσεις

- Όταν μια συνάρτηση δηλωθεί ως `virtual`, αυτό σημαίνει ότι η συνάρτηση μπορεί να γίνει `bound` σε πολλές διαφορετικές συναρτήσεις δυναμικά κατά το χρόνο εκτέλεσης.
  - Ο πολυμορφισμός αναφέρεται στη δυνατότητα να συσχετίσουμε πολλά νοήματα με μια (ιδεατή) συνάρτηση.
  - Παράδειγμα:
    - `class Student {public: virtual GradeReport();...};`
  - Η συνάρτηση `GradeReport()` μπορεί να αντιστοιχηθεί σε διαφορετικές συναρτήσεις στη παραγόμενες κλάσεις (το πρόγραμμα θα κοιτά στο αντικείμενο στο οποίο θα δείχνει ο δείκτης για να αποφασίσει ποια συνάρτηση θα κληθεί).

```
Student *sp1, *sp2, *sp3;  
Student s; Grad g, Undergrad u;  
sp1 = &s; sp2 = &g; sp3 = &u;  
sp1->GradeReport(); // εκτελείται η έκδοση του Student  
sp2->GradeReport(); // εκτελείται η έκδοση του Grad  
sp3->GradeReport(); // εκτελείται η έκδοση του Undergrad
```

# Η λύση στο αρχικό πρόβλημα

- Δημιουργία μιας ετερογενούς λίστας, χρήση ιδεατής συνάρτησης για να επιτευχθεί πρόσβαση στις διαφορετικές `GradeReport()` συναρτήσεις που βρίσκονται στις διαφορετικές παραγόμενες κλάσεις.

```
class Student {public: virtual GradeReport();...};  
...  
Student *list[3000];  
...  
for (int i=0;i<size;i++)  
    list[i]->GradeReport();
```

# Καθαρές ιδεατές συναρτήσεις (pure virtual functions)

- Οι κανονικές συναρτήσεις μέλη, συμπεριλαμβανομένων και των ιδεατών συναρτήσεων θα πρέπει να έχουν κάποια υλοποίηση.
  - Συνεπώς η ιδεατή συνάρτηση `GradeReport()` θα πρέπει κανονικά να έχει κάποια υλοποίηση στη βασική κλάση.
  - Ωστόσο, αυτό δεν είναι πάντα απαραίτητο (ή εφικτό). Μπορεί να μην υπάρχουν πολλά που μπορούν να γίνουν στις ιδεατές συναρτήσεις των βασικών κλάσεων καθώς η όποια λειτουργικότητα θα πρέπει να υλοποιηθεί στις παραγόμενες κλάσεις όπου θα υπάρχουν και περισσότερες πληροφορίες.
  - Η C++ επιτρέπει σε μια ιδεατή συνάρτηση να μην έχει υλοποίηση και σε αυτή τη περίπτωση η συνάρτηση λέγεται ότι είναι `pure virtual`, η δε υλοποίηση γίνεται στην παραγόμενη κλάση.
  - Μια ιδεατή συνάρτηση δηλώνεται ως `pure` τοποθετώντας το `=0` στη δήλωσή της.
    - `virtual void GradeReport() = 0;`



# Καθαρές ιδεατές συναρτήσεις και κλάσεις

- Όταν ένα αντικείμενο δηλώνεται ως αντικείμενο ενός κανονικού (concrete) τύπου κλάσης, γνωρίζουμε τα πάντα για το αντικείμενο (τα μέλη δεδομένων που διαθέτει, τη συμπεριφορά της κάθε συνάρτησης μέλους).
- Ωστόσο, αν ένα αντικείμενο δηλωθεί ως αντικείμενο μιας κλάσης που περιέχει μια καθαρή ιδεατή συνάρτηση τότε:
  - Το αντικείμενο δεν είναι καλά ορισμένο
  - Η συμπεριφορά της καθαρής ιδεατής συνάρτησης δεν είναι ορισμένη.
  - Ο μεταγλωττιστής δεν μπορεί να επιτρέψει την ύπαρξη τέτοιων αντικειμένων.
- Η C++ διαφοροποιεί τις κανονικές κλάσεις (concrete classes) από τις κλάσεις που περιέχουν καθαρές ιδεατές συναρτήσεις και οι οποίες κλάσεις ονομάζονται αφηρημένες κλάσεις (abstract classes).

# Αφηρημένες κλάσεις

- Μια αφηρημένη κλάση είναι μια κλάση με τουλάχιστον μια καθαρή ιδεατή συνάρτηση.
  - Μια αφηρημένη κλάση δεν μπορεί να δημιουργεί στιγμιότυπα.
    - `Student st; // error, Student is an abstract class, the object cannot be created`
  - Μια αφηρημένη κλάση μπορεί να χρησιμοποιηθεί για να δηλωθούν δείκτες
    - Ένας δείκτης δεν απαιτεί τη δημιουργία ενός αντικειμένου
    - Ένας δείκτης σε μια βασική κλάση μπορεί να δείχνει προς ένα αντικείμενο της παραγόμενης κλάσης.
    - `Student *st; // έγκυρο, μπορεί να δείχνει προς αντικείμενο της παραγόμενης κλάσης`

<https://github.com/chgogos/oop/blob/master/variou/COP3330/lect15/sample2.cpp>

<https://github.com/chgogos/oop/blob/master/variou/COP3330/lect15/sample3.cpp>

# Παράδειγμα

- Αφηρημένη κλάση Employee
- Κλάση Temporary, παραγόμενη κλάση από την Employee
- Κλάση Permanent, παραγόμενη κλάση από την Employee (αφηρημένη)
- Κλάση Hourly, παραγόμενη κλάση από την Permanent
- Κλάση Salaried, παραγόμενη κλάση από την Permanent

<https://github.com/chgogos/oop/tree/master/variuous/COP3330/lect15/employee>

# Ερωτήσεις σύνοψης

- Τι είναι μια ιδεατή συνάρτηση;
- Τι μπορούμε να πετύχουμε με τις ιδεατές συναρτήσεις;
- Πως ορίζεται μια καθαρή ιδεατή συνάρτηση;
- Τι είναι μια αφηρημένη κλάση;
- Μπορεί μια μεταβλητή να δηλωθεί ότι έχει ως τύπο, τον τύπο μιας αφηρημένης κλάσης;
- Μπορεί ένας δείκτης να δηλωθεί ότι έχει ως τύπο, τον τύπο μιας αφηρημένης κλάσης;

# Αναφορές

- <http://www.cs.fsu.edu/~xyuan/cop3330/>