Πολυμορφισμός και ιδεατές συναρτήσεις

#15

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

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 (GradeReport).
  - ```
    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 έκθεση προόδου 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.
    - `Grad gs; Undergrad 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.
    - `Grad gs; Student &rs = gs; Undergrad 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 Grad που έχει ήδη οριστεί
    `list[1] = &us; //` χρησιμοποιώντας ένα αντικείμενο Undergrad που έχει ήδη οριστεί
    `list[2] = new Grad; //` δυναμικά δεσμεύοντας μνήμη για ένα αντικείμενο Grad

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 περίσταση.
  - 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 void GradeReport();...};`
  - Η συνάρτηση `GradeReport()` μπορεί να αντιστοιχηθεί σε διαφορετικές συναρτήσεις στις 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
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect15/sample1.cpp

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 void GradeReport();…};
…
Student *list[3000];
   …
   for (int i=0;i<size;i++)
         list[i]->GradeReport();
```

Άσκηση #1: Ιδεατές συναρτήσεις

- Να υλοποιηθεί μια ιεραρχία κλάσεων για χαρακτήρες ενός video game, όπου η βασική κλάση Character θα διαθέτει την εικονική συνάρτηση attack() που εμφανίζει το μήνυμα "Character performs a basic attack!". Να υλοποιηθούν οι παράγωγες κλάσεις Warrior, Mage και Archer, καθεμία από τις οποίες θα παρακάπτει (override) τη συνάρτηση attack() εμφανίζοντας αντίστοιχα τα μηνύματα "Warrior swings a sword!", "Mage casts a spell!" και "Archer fires an arrow!".
- Στη main() να δημιουργηθεί ένας πίνακας δεικτών σε Character που θα περιέχει αντικείμενα των παραγώγων κλάσεων και να κληθεί η attack() για κάθε στοιχείο ώστε να εμφανίζεται το σωστό μήνυμα ανάλογα με τον πραγματικό τύπο του αντικειμένου.

Άσκηση #1: Λύση

```
class Character {
public:
    virtual void attack() const {
        cout << "Character performs a basic attack!" << endl;
    }
    virtual ~Character() {}
};

class Warrior : public Character {
public:
    void attack() const override {
        cout << "Warrior swings a sword!" << endl;
    }
};

class Mage : public Character {
public:
    void attack() const override {
        cout << "Mage casts a spell!" << endl;
    }
};

class Archer : public Character {
public:
    void attack() const override {
        cout << "Archer fires an arrow!" << endl;
    }
};
```

```
int main() {
    Character* characters[3];
    characters[0] = new Warrior();
    characters[1] = new Mage();
    characters[2] = new Archer();

    for (int i = 0; i < 3; ++i) {
        characters[i]->attack();
    }

    for (int i = 0; i < 3; ++i) {
        delete characters[i];
    }

    return 0;
}
```

```
Warrior swings a sword!
Mage casts a spell!
Archer fires an arrow!
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect15/exercise1.cpp

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 (abstract).

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/various/COP3330/lect15/sample2.cpp

https://github.com/chgogos/oop/blob/master/various/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/various/COP3330/lect15/employee

Άσκηση #2: Αφηρημένη κλάση

- Να υλοποιηθεί μια ιεραρχία κλάσεων για τον υπολογισμό κόστους διαφορετικών τύπων μεταφορών. Η βασική κλάση Transport να είναι αφηρημένη και να διαθέτει ένα protected μέλος δεδομένων distance (η απόσταση της διαδρομής σε χιλιόμετρα), το οποίο θα αρχικοποιείται μέσω κατασκευαστή. Η κλάση να διαθέτει επίσης μια καθαρή εικονική συνάρτηση cost() που θα επιστρέφει το κόστος μεταφοράς ως αριθμό κινητής υποδιαστολής.
- Να υλοποιηθούν οι παράγωγες κλάσεις Taxi, Bus και Train:
  - η Taxi θα υπολογίζει το κόστος ως base_fare + rate_per_km * distance,
  - η Bus θα επιστρέφει σταθερό εισιτήριο αξίας 1.40 ανεξαρτήτα από την τιμή του distance,
  - η Train θα υπολογίζει το κόστος ως distance * 0.05 + 2.0.
- Στη main() να δημιουργηθεί ένας πίνακας δεικτών τύπου Transport* με αντικείμενα των παραγώγων κλάσεων και να εμφανιστεί το κόστος κάθε μεταφοράς, αξιοποιώντας dynamic dispatching για να κληθεί η σωστή υλοποίηση της cost().

Άσκηση #2: Λύση

```
class Transport {
protected:
    double distance;
public:
    Transport(double d) : distance(d) {}
    virtual double cost() const = 0;
    virtual ~Transport() {}
};
class Taxi : public Transport {
    double base_fare;
    double rate_per_km;
public:
    Taxi(double d, double base, double rate)
        : Transport(d), base_fare(base), rate_per_km(rate) {}
    double cost() const override {
        return base_fare + rate_per_km * distance;
    }
};
class Bus : public Transport {
public:
    Bus(double d) : Transport(d) {}
    double cost() const override { return 1.40; }
};
class Train : public Transport {
public:
    Train(double d) : Transport(d) {}
    double cost() const override { return distance * 0.05 + 2.0; }
};
```

```
int main() {
    Transport* t[3];
    t[0] = new Taxi(10.0, 3.0, 0.8);
    t[1] = new Bus(25.0);
    t[2] = new Train(50.0);
    for (int i = 0; i < 3; ++i) {
        cout << "Cost: " << t[i]->cost() << endl;
    }

    for (int i = 0; i < 3; ++i) { delete t[i]; }

    return 0;
}
```

```
Cost: 11
Cost: 1.4
Cost: 4.5
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect15/exercise2.cpp

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;

Απαντήσεις στις ερωτήσεις σύνοψης

- Τι είναι μια ιδεατή συνάρτηση;
  - Ιδεατή συνάρτηση (virtual function) είναι μια συνάρτηση σε βασική κλάση που δηλώνεται ως virtual και προορίζεται να υπερκαλυφθεί (override) από τις παράγωγες κλάσεις.
- Τι μπορούμε να πετύχουμε με τις ιδεατές συναρτήσεις;
  - Με τις ιδεατές συναρτήσεις επιτυγχάνουμε dynamic dispatching, δηλαδή όταν καλούμε μια συνάρτηση μέσω δείκτη/αναφοράς στη βασική κλάση, εκτελείται η κατάλληλη έκδοση της συνάρτησης στην παράγωγο κλάση.
- Πως ορίζεται μια καθαρή ιδεατή συνάρτηση;
  - Ορίζεται με σύνταξη της μορφής: virtual void f() = 0;
- Τι είναι μια αφηρημένη κλάση;
  - Είναι κλάση που έχει τουλάχιστον μία καθαρή ιδεατή συνάρτηση και από την οποία δεν μπορούν να δημιουργηθούν αντικείμενα.
- Μπορεί μια μεταβλητή να δηλωθεί ότι έχει ως τύπο, τον τύπο μιας αφηρημένης κλάσης;
  - Όχι, δεν μπορούμε να κατασκευάσουμε αντικείμενα αφηρημένης κλάσης.
- Μπορεί ένας δείκτης να δηλωθεί ότι έχει ως τύπο, τον τύπο μιας αφηρημένης κλάσης;
  - Ναι, μπορούμε να δηλώσουμε δείκτες ή αναφορές σε αφηρημένη κλάση και να τους αναθέσουμε αντικείμενα παραγώγων κλάσεων (βασικό στοιχείο του πολυμορφισμού).

Αναφορές

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