Αντικειμενοστραφής Προγραμματισμός

Μέρος 6ο: Υπερφόρτωση τελεστών, φίλιες συναρτήσεις

Εξάμηνο Σπουδών: 6ο
Κωδικός Μαθήματος: 647

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Περιεχόμενα

- Υπερφόρτωση τελεστών
  - Μονομελείς και διμελείς τελεστές
  - Υλοποίηση υπερφόρτωσης μέσω συναρτήσεων μελών

- Φίλιες συναρτήσεις και αυτόματη μετατροπή τύπων
  - Φίλιες συναρτήσεις και φίλιες κλάσεις
  - Κατασκευαστές για την αυτόματη μετατροπή τύπων
  - Υπερφόρτωση των τελεστών << και >>
  - Οι τελεστές: `=`, `[]`, `++`, `--`

Εισαγωγή

- Στην C++, οι τελεστές είναι συναρτήσεις.
  - Απλά καλούνται με διαφορετική σύνταξη.
  - Παραδείγματα τελεστών: `+`, `-`, `%`, `==` κ.ο.κ.
- Παράδειγμα εφαρμογής: `int y = x + 7;`
  - Το + είναι ένας διμελής τελεστής ενώ τα `x` και `7` είναι οι τελεστέοι στους οποίους εφαρμόζεται
  - Αντίστοιχα, το = είναι και αυτός ένας διμελής τελεστής ενώ τα `y` και `x + 7` είναι οι τελεστέοι
- Προτιμούμε αυτή τη σημειογραφία γιατί είμαστε πιο εξοικειωμένοι με αυτή.
  - Εναλλακτικά, κάποιος θα έγραφε: `=(y, +(x, 7))`
  - Όπου, π.χ., το + είναι το όνομα της συνάρτησης, ενώ τα `x` και `7` είναι τα ορίσματα της

Τα βασικά της υπερφόρτωσης τελεστών

- Η υπερφόρτωση τελεστή είναι διαδικασία όμοια με αυτή της υπερφόρτωσης συνάρτησης
  - ουσιαστικά, ο τελεστής είναι το “όνομα” της συνάρτησης

- Στο παράδειγμα, γίνεται υπερφόρτωση του τελεστή +
  - για την περίπτωση που οι τελεστέοι είναι αντικείμενα τύπου `Money`
  - επιτρέποντας έτσι την πρόσθεση αντικειμένων τύπου `Money`
  - οι παράμετροι είναι `const` με αναφορά για λόγους αποδοτικότητας

```
const Money operator+(const Money& amount1, const Money& amount2) {
    int allCents1 = amount1.getCents() + amount1.getEuros() * 100;
    int allCents2 = amount2.getCents() + amount2.getEuros() * 100;
    int sumAllCents = allCents1 + allCents2;
    int finalEuros = abs(sumAllCents) / 100;
    int finalCents = abs(sumAllCents) % 100;
    if (sumAllCents < 0) //Money can be negative
        return Money(-finalEuros, -finalCents);
    return Money(finalEuros, finalCents);
}
```

Τα βασικά της υπερφόρτωσης τελεστών

- Παρατηρήστε ότι η συνάρτηση `operator+` δεν είναι μέλος της κλάσης `Money`
  - η υλοποίηση της είναι “λίγο πιο πολύπλοκη” από αυτή της συνάρτησης μέλους “`add`”
  - π.χ., απαιτεί ειδικό χειρισμό θετικών/αρνητικών τιμών

- Γενικά, ωστόσο, η υπερφόρτωση του τελεστή + δεν είναι δύσκολη διαδικασία
  - απλά εκτελεί την πράξη της πρόσθεσης σε τύπους που ορίζονται από το χρήστη
  - και παράλληλα βελτιώνει τον παραγόμενο κώδικα

```
const Money operator+(const Money& amount1, const Money& amount2) {
    int allCents1 = amount1.getCents() + amount1.getEuros() * 100;
    int allCents2 = amount2.getCents() + amount2.getEuros() * 100;
    int sumAllCents = allCents1 + allCents2;
    int finalEuros = abs(sumAllCents) / 100;
    int finalCents = abs(sumAllCents) % 100;
    if (sumAllCents < 0) //Money can be negative
        return Money(-finalEuros, -finalCents);
    return Money(finalEuros, finalCents);
}
```

Μια παρατήρηση για τους κατασκευαστές

- Ερώτημα: Είναι ο κατασκευαστής `void` συνάρτηση?
  - Απάντηση: Έτσι “αντιλαμβανόμαστε” τον κατασκευαστή, αλλά όχι

- Ο κατασκευαστής είναι μια ειδική περίπτωση συνάρτησης
  - με ιδιαίτερες ιδιότητες
  - και βέβαια μπορεί να επιστρέψει τιμή

- Παρατηρήστε την τιμή επιστροφής στην υπερφόρτωση του τελεστή + στο παράδειγμα:
  `return Money(finalEuros, finalCents);`
  - πρόκειται για ένα αντικείμενο της κλάσης `Money`
  - ο κατασκευαστής ”επιστρέφει” ένα αντικείμενο, το οποίο ονομάζεται ανώνυμο (anonymous)

Επιστροφή `const` τιμής

- Παρατηρήστε ξανά την υπερφόρτωση του τελεστή +
  - Ερώτημα: Γιατί επιστρέφεται `const` αντικείμενο;
- Ας εξετάσουμε το σενάριο της επιστροφής ενός αντικειμένου που δεν είναι `const`
  - σε αυτή την περίπτωση η κλήση `(m1+m2).input()` τροποποιεί (με δεδομένα από την είσοδο) το ανώνυμο αντικείμενο `m1+m2`
  - αυτό το αποτρέπουμε, ορίζοντας `const` το αντικείμενο επιστροφής

```
const Money operator+(const Money& amount1, const Money& amount2);

int main() {
  Money m1, m2;
  (m1+m2).output();  //Legal: This does not modify the anonymous object m1+m2
  (m1+m2).input();   //Illegal: The anonymous object m1+m2 must not be modified

  return 0;
}
```

Υπερφόρτωση του τελεστή ==

- Ο τελεστής == επιτρέπει τη σύγκριση αντικειμένων
- Ο τύπος επιστροφής είναι `bool`:
  - `true`, όταν ισχύει η ισότητα
  - `false`, διαφορετικά
- Παρατήρηση: Και πάλι δεν πρόκειται για συνάρτηση μέλος
  - όπως και στην υπερφόρτωση του τελεστή +

```
//Declaration
bool operator==(const Money& amount1, const Money& amount2);

//Implementation
bool operator==(const Money& amount1, const Money& amount2) {
    return ((amount1.getEuros() == amount2.getEuros())
         && (amount1.getCents() == amount2.getCents()));
}
```

Υπερφόρτωση του τελεστή -

- Ο τελεστής - μπορεί να υπερφορτωθεί με δύο τρόπους:
  - ως διμελής τελεστής (αφαίρεση)
  - ως μονομελής τελεστής (άρνηση)

- Προτείνεται η υποστήριξη υλοποιήσεων και για τους δύο
  - Κατά την υπερφόρτωση του μονομελούς τελεστή δίνεται μόνο ένα όρισμα
  - Στην υπερφόρτωση του διμελούς τελεστή δίνονται δύο ορίσματα

```
//Declarations
const Money operator -(const Money& amount);
const Money operator -(const Money& amount1, const Money& amount2);

//Implementations
const Money operator-(const Money& amount) {
    return Money(-amount.getEuros(), -amount.getCents());
}
const Money operator-(const Money& amount1, const Money& amount2) {
    return amount1 + (-amount2); //<-- Call to unary operator - and to binary operator +
}
```

Υπερφόρτωση μέσω συναρτήσεων μελών

- Η υπερφόρτωση των τελεστών που είδαμε έως τώρα έγινε μέσω τυπικών συναρτήσεων
- Εναλλακτικά, ωστόσο, η υπερφόρτωση μπορεί να γίνει και μέσω συναρτήσεων μελών
- Όταν ένας διμελής τελεστής ορίζεται μέσω μιας συνάρτησης μέλους:
  - τότε στην υλοποίηση του δέχεται μόνο μια παράμετρο, όχι δύο
  - ο πρώτος τελεστέος είναι το αντικείμενο από το οποίο γίνεται η κλήση
  - ο δεύτερος τελεστέος είναι το αντικείμενο παράμετρος της συνάρτησης

```
const Money Money::operator-() const {
    return Money(-euros, -cents);
}
const Money Money::operator-(const Money& secondOperand) const {
    return *this + (-secondOperand); //<-- Call to unary operator - and to binary operator +
}
bool Money::operator==(const Money& secondOperand) const {
    return ((euros == secondOperand.euros) && (cents == secondOperand.cents));
}
```

Επεξήγηση με ένα παράδειγμα

- Θεωρήστε τον παρακάτω κώδικα και υποθέστε ότι η υπερφόρτωση του τελεστή + έχει γίνει μέσω συνάρτησης μέλους
- Τότε:
  - το αντικείμενο `cost` καλεί τη συνάρτηση μέλος +
  - το αντικείμενο `tax` είναι η παράμετρος στην κλήση
  - η κλήση δηλαδή γίνεται κάπως έτσι: `total = cost.+(tax);`

```
class Money {
public:
  const Money operator+(const Money& amount);
  ...
}
int main() {
  Money  cost(1, 50), tax(0, 15), total;
  total = cost + tax;
  return 0;
}
```

Hands-on ενότητα 1

Υλοποίηση της κλάσης `Money`

Ποια είναι η προτεινόμενη μέθοδος υπερφόρτωσης τελεστών;

- Στο πλαίσιο των γενικών αρχών του αντικειμενοστρεφούς προγραμματισμούς προτείνεται
  - η μέθοδος υπερφόρτωσης μέσω συναρτήσεων μελών
  - για τη διατήρηση του “πνεύματος” του αντικειμενοστρεφούς προγραμματισμούς
- Η μέθοδος αυτή είναι και πιο αποτελεσματική καθώς
  - δεν απαιτεί την κλήση συναρτήσεων μελών
  - setter ή getter
- Ωστόσο, υπάρχει (τουλάχιστον) ένα σημαντικό μειονέκτημα
  - όπως θα δούμε παρακάτω...

Υπερφόρτωση του τελεστή κλήσης συνάρτησης ()

- Ο τελεστής κλήσης συνάρτησης (function call operator):
  - πρέπει να υπερφορτωθεί ως συνάρτηση-μέλος
  - επιτρέπει τη “χρήση αντικειμένων ως συναρτήσεις”
  - μπορεί να υπερφορτωθεί για διαφορετικούς τύπους ορισμάτων

- Στο παρακάτω παράδειγμα:
  - ο τελεστής `()` έχει υπερφορτωθεί για ζεύγη ακεραίων

```
class Money {
public:
  Money& operator()(int theEuros, int theCents);
  ...
}
Money& Money::operator()(int theEuros, int theCents) {
  euros = theEuros;
  cents = theCents;
  return *this;
}
```

Υπερφόρτωση άλλων τελεστών

- Τυπικά, οι τελεστές `&&` και `||` εφαρμόζονται σε `bool` τύπους
- Υπενθυμίζεται ότι οι τελεστές αυτοί χρησιμοποιούν την αξιολόγηση McCarthy
  - σύμφωνα με την οποία το δεύτερο όρισμα αξιολογείται μόνο εάν το πρώτο όρισμα δεν επαρκεί για τον προσδιορισμό της τιμής της έκφρασης.
- Όταν γίνεται υπερφόρτωση τους δεν χρησιμοποιείται πλέον η αξιολόγηση McCarthy
  - χρησιμοποιείται η “πλήρης εκτίμηση”
  - αντίθετα με τι θα προσδοκούσε κάποιος
- Γενικώς, θα πρέπει να αποφεύγεται η υπερφόρτωση των τελεστών αυτών

Συναρτήσεις const

- Πότε μια συνάρτηση ορίζεται `const`;
  - Οι `const` συναρτήσεις μέλη δεν επιτρέπεται να αλλάξουν τις τιμές των μεταβλητών μελών της κλάσης
  - Τα αντικείμενα τα οποία έχουν δηλωθεί `const` κατά τη δημιουργία τους μπορούν να καλέσουν μόνο `const` συναρτήσεις μέλη

- Καλή πρακτική: Οι συναρτήσεις μέλη που δεν τροποποιούν τις τιμές των μεταβλητών μελών της κλάσης θα πρέπει να ορίζονται `const`
  - Αυτό γίνεται με τη χρήση της λέξης-κλειδί `const` στο τέλος της υπογραφής τους

```
class Distance {
private:
    int yards, feet;
public:
    int getYards() const;
    int getFeet() const;
};
```

```
int main() {
    const usFootballField(3600, 0);
    cout << "An American football field is "
         << usFootballField.getYards()
         << " long." << endl;
    return 0;
}
```

Φίλιες συναρτήσεις

- Οι φίλιες συναρτήσεις δεν είναι συναρτήσεις μέλη
- Οι φίλιες συναρτήσεις μια κλάσης έχουν πρόσβαση στα ιδιωτικά μέλη της κλάσης αυτής
  - όπως ακριβώς έχουν και οι συναρτήσεις μέλη της κλάσης αυτής
- Οι φίλιες συναρτήσεις μια κλάσης δηλώνονται στην κλάση χρησιμοποιώντας τη λέξη-κλειδί `friend`

```
class Box {
private:
   double width;
   double height;
public:
   //Member function
   void setDimensions(double w, double h);
   //Friend function
   friend void printDimensions(Box box);
};
```

```
//Member function definition
void Box::setDimensions(double w, double h) {
  width = w;
  height = h;
}
//Note: printDimensions() is not a member function
//As a friend of Box, it can access any member of it
void printDimensions(Box box) {
  cout << "Box " << box.width << "x" << box.height;
}
```

Φίλιες συναρτήσεις

- Οι φίλιες συναρτήσεις είναι εξαιρετικά χρήσιμες σε αρκετές περιπτώσεις
- Παράδειγμα: Η υπερφόρτωση ενός τελεστή μέσω τυπικής συνάρτησης (σ.σ., όχι μέλους)
  - σε μια τέτοια περίπτωση, η πρόσβαση στα ιδιωτικά πεδία της κλάσης γίνεται αναγκαστικά με κλήση κατάλληλων συναρτήσεων μελών (setter ή getter)
  - αναποτελεσματική προσέγγιση λόγω του κόστους των κλήσεων των συναρτήσεων
- Καλή πρακτική: Ορίζουμε ως `friend` τη συνάρτηση που υλοποιεί την υπερφόρτωση. Έτσι αποκτά άμεση πρόσβαση στα ιδιωτικά μέλη.

```
class Money {
public:
    friend const Money operator-(const Money& amount);
    ...
};
const Money operator-(const Money& amount) {
    return Money(-amount.euros, -amount.cents); // <-- euros and cents are private
}                                               //     members of class Money
```

Φίλιες συναρτήσεις και υπερφόρτωση τελεστών

- Η υπερφόρτωση τελεστών αποτελεί την πιο συνήθη περίπτωση χρήσης φίλιων συναρτήσεων
- Βελτιώνει την αποδοτικότητα καθώς αποφεύγεται η κλήση συναρτήσεων μελών (getter ή setter)
- Δεν υπάρχει σαφής τρόπος καθορισμού των φίλιων συναρτήσεων μιας κλάσης
- Οποιαδήποτε συνάρτηση μπορεί να εμπίπτει σε αυτή την κατηγορία
- Υπάρχουν όμως πλεονεκτήματα;
  - Ιδιαίτερα στην υπερφόρτωση των τελεστών, πάρα πολλά
  - Επιτρέπουν την αυτόματη μετατροπή τύπων
  - Υποστηρίζουν την ενθυλάκωση: οι φίλιες συναρτήσεις αποτελούν μέρος της κλάσης
  - Βελτιώνουν την αποδοτικότητα

Οι φίλιες συναρτήσεις στον αντικειμενοστρεφή προγραμματισμό

- Ερώτημα: Είναι οι φίλιες συναρτήσεις μέρος του αντικειμενοστρεφή προγραμματισμού ή θα πρέπει να αποφεύγονται;
  - Γενικά, πιστεύεται ότι οι φίλιες συναρτήσεις παραβιάζουν τις βασικές αρχές του αντικειμενοστρεφούς προγραμματισμού (αν και πλεονεκτούν όπως είδαμε)
  - Για την υπερφόρτωση των τελεστών συγκεκριμένα, συνίσταται η υλοποίηση μέσω συναρτήσεων μελών της κλάσης

Φίλιες κλάσεις

- Όπως οι συναρτήσεις, έτσι και οι κλάσεις μπορεί να είναι φίλιες
- Παράδειγμα: Η κλάση `List` είναι φίλη της κλάσης `Node`
  - όλα τα μέλη της κλάσης `Node` είναι προσβάσιμα από την κλάση `List`
  - όχι όμως και αντιστρόφως
- Η δήλωση ότι μια κλάση είναι φίλη μιας άλλης κλάσης γίνεται στην κλάση "εξουσιοδότησης"
  - στο παράδειγμα, στην κλάση `Node`

```
template <typename E>
class List {
private:
    Node<E> *head;
public:
    E& front() const {
        return head->elem;
    }
};
```

```
template <typename E>
class Node {
private:
    E value;
    Node* next;
    Node() : next(NULL) {} //Private constructor
    //List is declared as friend of Node class
    friend class List;
};
```

Υπερφόρτωση των τελεστών εισόδου εξόδου (>> και <<)

- Η υπερφόρτωση αυτών των τελεστών επιτρέπει την είσοδο και έξοδο αντικείμενων
- Γίνεται όμοια με την υπερφόρτωση των υπολοίπων τελεστών
- Βελτιώνει την αναγνωσιμότητα του κώδικα (όπως κάνουν και οι υπόλοιποι τελεστές)
- Επιτρέπει την ανάπτυξη κώδικα όπως παρακάτω:
  - `cin >> anObejct;`
  - `cout << anObejct;`
- Αντί της κλασικής προσέγγισης μέσω μεθόδων:
  - `myObject.input();`
  - `myObject.output();`

Υπερφόρτωση του τελεστή εξόδου <<

- Ο τελεστής εξόδου << συνήθως χρησιμοποιείται με το αντικείμενο `cout`
- Αποτελεί ένα διμελή τελεστή
- Παράδειγμα: `cout << "Hello";`
  - Ο πρώτος τελεστέος είναι το αντικείμενο `cout`
    - της κλάσης `ostream`
    - ως μέρος της βιβλιοθήκης `iostream`
  - Ο δεύτερος τελεστέος είναι η συμβολοσειρά ”Hello”
    - Γενικά, ο δεύτερος τελεστέος δύναται να είναι αντικείμενο μιας κλάσης που έχουμε υλοποιήσει εμείς
    - Π.χ. ένα αντικείμενο της κλάσης `Money`

Υπερφόρτωση του τελεστή εξόδου <<

- Θεωρήστε το παρακάτω παράδειγμα και παρατηρήστε ότι για την υποστήριξη διαδοχικών κλήσεων, ο τελεστής εξόδου << πρέπει να επιστρέψει κάποια τιμή
- Ερώτημα: Ποιος είναι ο τύπος επιστροφής;
  - Απάντηση: το αντικείμενο `cout` (στο παράδειγμα).
  - Γενικά: ένα αντικείμενο τύπου `ostream`
- Αντίστοιχα, για τον τελεστή εισόδου:
  - Ο πρώτος τελεστέος και ο τύπος επιστροφής είναι αντικείμενα της κλάσης `istream`
  - Ο δεύτερος τελεστέος μπορεί να είναι αντικείμενο κλάσης που έχουμε υλοποιήσει εμείς

```
int main() {
    Money amount(100);
    cout << "I have " << amount << endl;
}
```

Παράδειγμα υλοποίησης

```
class Money {
public:
    friend ostream& operator <<(ostream& outputStream, const Money& amount);
    friend istream& operator >>(istream& inputStream, Money& amount);
};
```

```
ostream& operator <<(ostream& outputStream, const Money& amount) {
    if (amount.euros < 0 || amount.cents < 0)
        outputStream << "-";
    outputStream << abs(amount.euros) << '.';
    if (abs(amount.cents) < 10)
        outputStream << '0';
    outputStream << abs(amount.cents) << " euros";
    return outputStream;
}
```

```
istream& operator >>(istream& inputStream, Money& amount) {
    double amountAsDouble;
    inputStream >> amountAsDouble;
    amount.euros = amount.eurosPart(amountAsDouble);
    amount.cents = amount.centsPart(amountAsDouble);
    return inputStream;
}
```

Hands-on ενότητα 2

Επέκταση της κλάσης `Money` για την υποστήριξη των τελεστών εισόδου και εξόδου

Υπερφόρτωση του τελεστή ανάθεσης =

- Ο τελεστής ανάθεσης πρέπει να υπερφορτωθεί ως συνάρτηση-μέλος
  - αν δε γίνει αυτό, υπερφορτώνεται αυτόματα
- Η προκαθορισμένη υλοποίηση εκτελεί αντιγραφή μέλος-προς-μέλος
  - οι μεταβλητές-μέλη ενός αντικειμένου αντιγράφονται στις αντίστοιχες μεταβλητές-μέλη του άλλου αντικειμένου
  - συνήθως αρκεί για απλές κλάσεις
- Υπερφόρτωση του τελεστή ανάθεσης απαιτείται σε περιπτώσεις που υπάρχουν δείκτες

Οι τελεστές προσαύξησης και μείωσης

- Καθένας από αυτούς τους δύο τελεστές έχει δύο εκδόσεις:
  - Προθεματική έκδοση: `++x;`
  - Μεταθεματική έκδοση: `x++;`
- Στην υλοποίηση πρέπει να γίνει αυτή η διάκριση
- Τυπικά υπερφορτώνεται η προθεματική έκδοση
- Για την υπερφόρτωση της μεταθεματικής έκδοσης:
  - απαιτείται μια ακόμη παράμετρος τύπου `int`
  - η οποία δεν χρησιμοποιείται κάπου
  - είναι απλά μια υπόδειξη για τον compiler

Σύνοψη

- Οι προκαθορισμένοι τελεστές της C++ υπερφορτώνονται για να λειτουργήσουν για τις δικές μας κλάσεις
- Οι τελεστές είναι απλά συναρτήσεις
- Οι φίλιες συναρτήσεις έχουν άμεση πρόσβαση στα ιδιωτικά μέλη των κλάσεων
- Οι τελεστές μπορούν να υπερφορτωθούν και ως συναρτήσεις μέλη
  - Σε αυτή την περίπτωση, ο πρώτος τελεστέος είναι το αντικείμενο το οποίο κάνει την κλήση