

# Η δεσμευμένη λέξη friend, κατασκευαστές μετατροπής, καταστροφείς

#4

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Ιωαννίνων (Άρτα)

Γκόγκος Χρήστος

# Παράδειγμα κίνητρο (1/3)

- Έστω ότι επιθυμούμε να γράψουμε μια συνάρτηση Equals που να συγκρίνει 2 αντικείμενα Fraction και να τη χρησιμοποιήσουμε όπως στη συνέχεια:

```
int main() {  
    Fraction f1(1, 2);  
    Fraction f2(2, 4);  
    if (Equals(f1, f2))  
        cout << "The fractions are equal" << endl;  
    return 0;  
}
```

- Μια πιθανή υλοποίηση της συνάρτησης Equals είναι η ακόλουθη:

```
bool Equals(Fraction x, Fraction y) {  
    if (x.GetNumerator() * y.GetDenominator() == y.GetNumerator() * x.GetDenominator())  
        return true;  
    else  
        return false;  
}
```

# Παράδειγμα κίνητρο (2/3)

- Τι θα συνέβαινε αν επιθυμούμε να γράψουμε τη συνάρτηση `Equals` ως εξής:

```
bool Equals(Fraction x, Fraction y) {  
    if (x.numer * y.denom ==  
        y.numer * x.denom)  
        return true;  
    else  
        return false;  
}
```

Καθώς η συνάρτηση `Equals` δεν είναι συνάρτηση μέλος της `Fraction`, δεν μπορεί να προσπελαύνει απευθείας τις ιδιωτικές συναρτήσεις μέλη.

```
$ g++ Fraction.cpp main2.cpp -o main  
main2.cpp: In function 'bool Equals(Fraction, Fraction)':  
main2.cpp:8:11: error: 'int Fraction::numer' is private within this context  
    if (x.numer * y.denom == y.numer * x.denom)  
        ^~~~~  
In file included from main2.cpp:2:  
Fraction.hpp:14:9: note: declared private here  
    int numer;  
        ^~~~~  
main2.cpp:8:21: error: 'int Fraction::denom' is private within this context  
    if (x.numer * y.denom == y.numer * x.denom)  
        ^~~~~  
In file included from main2.cpp:2:  
Fraction.hpp:15:9: note: declared private here  
    int denom;  
        ^~~~~  
main2.cpp:8:32: error: 'int Fraction::numer' is private within this context  
    if (x.numer * y.denom == y.numer * x.denom)  
        ^~~~~  
In file included from main2.cpp:2:  
Fraction.hpp:14:9: note: declared private here  
    int numer;  
        ^~~~~  
main2.cpp:8:42: error: 'int Fraction::denom' is private within this context  
    if (x.numer * y.denom == y.numer * x.denom)  
        ^~~~~  
In file included from main2.cpp:2:  
Fraction.hpp:15:9: note: declared private here  
    int denom;  
        ^~~~~
```

## Παράδειγμα κίνητρο (3/3)

- Ωστόσο, συναρτήσεις όπως η `Equals` ενδέχεται να χρειάζεται να προσπευλεύουν ιδιωτικά μέλη συχνά, οπότε θα θέλαμε να έχουμε τη δυνατότητα να προσπελεύουμε απευθείας ιδιωτικά μέλη της κλάσης.

<https://github.com/chgogos/oop/tree/master/variou/COP3330/lect4/sample1>

- Μια λύση στη C++ είναι η χρήση της δεσμευμένης λέξης `friend`.

# Η δεσμευμένη λέξη friend

- Η δεσμευμένη λέξη friend επιτρέπει σε μια κλάση να χορηγήσει **πλήρη πρόσβαση** σε μια **εξωτερική οντότητα**.
  - Πλήρης πρόσβαση σημαίνει δυνατότητα προσπέλασης και των ιδιωτικών μελών της κλάσης.
  - Εξωτερική οντότητα μπορεί να είναι μια συνάρτηση, ή ακόμα και μια άλλη κλάση.
- Για να αποδοθεί η ιδιότητα friend, θα πρέπει να προστεθεί η δήλωση friend μέσα στη δήλωση της κλάσης, και να ακολουθήσει η αντίστοιχη οντότητα.
  - Μια friend οντότητα δεν είναι ούτε δημόσια ούτε ιδιωτική, καθώς δεν πρόκειται για μέλος της κλάσης, οπότε δεν έχει σημασία αν θα τοποθετηθεί σε τμήμα public ή private της κλάσης.
  - Μια friend συνάρτηση σε μια κλάση έχει πλήρη πρόσβαση στα ιδιωτικά μέλη της κλάσης.

# Παράδειγμα με friend συναρτήσεις (1/2)

- Στο παράδειγμα που ακολουθεί:
  - Ορίζεται η friend συνάρτηση Equals()
  - Ορίζεται η friend συνάρτηση Add() που προσθέτει δύο κλάσματα και επιστρέφει το αποτέλεσμα.
  - Στο αρχείο main.cpp βρίσκεται το πρόγραμμα οδηγός.

[https://github.com/chgogos/oop/tree/master/various/COP3330/lect4/sample2\\_friend](https://github.com/chgogos/oop/tree/master/various/COP3330/lect4/sample2_friend)

# Παράδειγμα με friend συναρτήσεις (2/2)

```
class Fraction
{
    ...
    friend bool Equals(Fraction x, Fraction y);
    friend Fraction Add(Fraction x, Fraction y);
};
```

Fraction.hpp

```
...
bool Equals(Fraction x, Fraction y) {
    if (x.numer * y.denom == y.numer * x.denom)
        return true;
    else
        return false;
}
Fraction Add(Fraction x, Fraction y) {
    int num = x.numer * y.denom + y.numer * x.denom;
    int denom = x.denom * y.denom;
    Fraction answer(num, denom);
    return answer;
}
```

Fraction.cpp

```
#include <iostream>
#include "Fraction.hpp"
using namespace std;
int main() {
    Fraction f1(1, 2);
    Fraction f2(2, 4);
    if (Equals(f1, f2))
        cout << "The fractions are equal" << endl;
    Fraction f3 = Add(f1, f2);
    f3.Show();
    return 0;
}
```

main.cpp

```
$ g++ Fraction.cpp main.cpp -o main
$ ./main
The fractions are equal
8/8
```

# Χρήση συνάρτησης μέλους αντί για friend συνάρτηση

- Όταν μια συνάρτηση πραγματοποιεί επεξεργασία σε δύο αντικείμενα, συχνά είναι βολικό να περνάμε ως παραμέτρους και τα δύο αντικείμενα και να κάνουμε τη συνάρτηση friend.
- Μια άλλη επιλογή είναι να χρησιμοποιήσουμε μια συνάρτηση μέλους αλλά σε αυτή την περίπτωση ένα από τα αντικείμενα θα πρέπει να είναι το καλών αντικείμενο (calling object).
  - Παράδειγμα: Η συνάρτηση Equals() θα μπορούσε να οριστεί ως συνάρτηση μέλος της κλάσης Fraction, και να κληθεί ως εξής:

```
if (f1.Equals(f2))  
    cout << "The fractions are equal" << endl;
```
  - Στο παραπάνω κώδικα, f1 είναι το καλών αντικείμενο (δηλαδή το αντικείμενο που καλεί τη συνάρτηση μέλος) και το f2 περνά στη συνάρτηση Equals της f1 ως όρισμα.



# Παράδειγμα με συναρτήσεις μέλη αντί για friend συναρτήσεις

```
class Fraction {  
public:  
    ...  
    bool Equals(Fraction other);  
    Fraction Add(Fraction other);  
    ...  
};
```

Fraction.hpp

```
#include <iostream>  
#include "Fraction.hpp"  
using namespace std;  
int main() {  
    Fraction f1(1, 2);  
    Fraction f2(2, 4);  
    if (f1.Equals(f2))  
        cout << "The fractions are equal" << endl;  
    f2.SetValue(2, 7);  
    Fraction f3 = f1.Add(f2);  
    f3.Show();  
    cout << "Decimal value: " << f3.Evaluate() << endl;  
    return 0;  
}
```

main.cpp

```
bool Fraction::Equals(Fraction other) {  
    return numer * other.denom == other.numer * denom;  
}  
  
Fraction Fraction::Add(Fraction other) {  
    int n = numer * other.denom + other.numer * denom;  
    int d = denom * other.denom;  
    return Fraction(n, d);  
}
```

Fraction.cpp

The fractions are equal  
11/14  
Decimal value: 0.785714

[https://github.com/chgogos/oop/tree/master/various/COP3330/lect4/sample3\\_member](https://github.com/chgogos/oop/tree/master/various/COP3330/lect4/sample3_member)

# Συναρτήσεις μέλη vs. φίλες συναρτήσεις

- Η επιλογή του εάν θα χρησιμοποιηθεί συνάρτηση μέλος ή friend συνάρτηση είναι θέμα προγραμματιστικού στυλ.
- Διαφορετικοί προγραμματιστές μπορεί να έχουν διαφορετικές προτιμήσεις για το θέμα. Στη συνέχεια παρουσιάζονται για σύγκριση των δύο κλήσεων ο ένας κώδικας μετά τον άλλο:

```
f3 = Add(f1, f2); // κλήση στη friend συνάρτηση Add  
f3 = f1.Add(f2); // κλήση στη συνάρτηση μέλος Add της κλάσης Fraction
```
- Ωστόσο, αξίζει να σημειωθεί ότι οι δύο παραπάνω κλήσεις δεν είναι πάντα ισοδύναμες.
  - Στην έκδοση με τις friend συναρτήσεις η Add λαμβάνει αντίγραφα από τα f1 και f2 (συνεπώς η συνάρτηση δεν μπορεί να αλλάξει τα αρχικά αντικείμενα).
  - Στην έκδοση με τη συνάρτηση μέλος, η συνάρτηση Add μπορεί να αλλάξει το αντικείμενο f1.

# Κατασκευαστές μετατροπής

- Ορισμένοι ενσωματωμένοι τύποι μπορούν να πραγματοποιήσουν αυτόματη μετατροπή τύπων όπως:

```
int x = 5;  
double y = 4.5, z = 1.2;  
y = x; // έγκυρο, αυτόματη μετατροπή  
z = x + y; // έγκυρο, αυτόματη μετατροπή
```

- Παρόμοια λειτουργικότητα μπορεί να προστεθεί και σε κλάσεις που δημιουργούμε χρησιμοποιώντας τους λεγόμενους **κατασκευαστές μετατροπής** (conversion constructors).

# Κατασκευαστές μετατροπής

- Ένας κατασκευαστής μετατροπής είναι ένας κατασκευαστής με μια παράμετρο.
  - Καθώς ο κατασκευαστής δημιουργεί/αρχικοποιεί ένα νέο αντικείμενο, μπορούμε να χρησιμοποιήσουμε έναν κατασκευαστή μετατροπής για να μετατρέψουμε μια μεταβλητή με τύπο τον τύπο της παραμέτρου σε ένα νέο αντικείμενο.

- Παράδειγμα κατασκευαστή μετατροπής:

```
Fraction(int n); /* μπορεί να χρησιμοποιηθεί για να μετατρέψει έναν ακέραιο σε  
                Fraction, αρχικοποιεί το κλάσμα σε n/1 */
```

- Ο παραπάνω κατασκευαστής μπορεί να χρησιμοποιηθεί για να πραγματοποιούνται αυτόματες μετατροπές τύπων όπως οι ακόλουθες:

```
Fraction f1, f2;  
f1 = Fraction(4) /* ρητή κλήση του κατασκευαστή. Δημιουργείται το  
                κλάσμα 4/1 και ανατίθεται στο f1 */  
f2 = 10;        /* υπονοούμενη κλήση του κατασκευαστή μετατροπής.  
                Ισοδύναμο με f2 = Fraction(10); */  
f1 = Add(f2, 5); /* ο κατασκευαστής μετατροπής μετατρέπει το 5 σε 5/1 */
```

# Κατασκευαστής μετατροπής

- Ένας κατασκευαστής με πολλές παραμέτρους μπορεί να είναι κατασκευαστής μετατροπής αν όλες, πλην μιας, από τις παραμέτρους είναι προαιρετικές (optional):  
`Fraction(int n, int d=1);`
- Η αυτόματη μετατροπή τύπων από κατασκευαστές μπορεί να αποτραπεί χρησιμοποιώντας τη δεσμευμένη λέξη **explicit** στην αρχή της δήλωσης:  
`explicit Fraction(int n);`
- Ο ανωτέρω κατασκευαστής δεν θα πραγματοποιεί πλέον αυτόματη μετατροπή ακεραίων σε `Fraction`.

# Παράδειγμα με κατασκευαστή μετατροπής

```
class Fraction {  
public:  
    Fraction();  
    Fraction(int n, int d = 1);  
    ...  
};
```

Fraction.hpp

```
...  
Fraction::Fraction() {  
    cout << "Fraction()" << endl;  
    numer = 0;  
    denom = 1;  
}  
Fraction::Fraction(int n, int d) {  
    cout << "Fraction(int, int)" << endl;  
    numer = n;  
    denom = d;  
}  
...
```

Fraction.cpp

```
#include <iostream>  
#include "Fraction.hpp"  
using namespace std;  
int main() {  
    // ρητές κλήσεις κατασκευαστών  
    Fraction f1;  
    f1.Show();  
    Fraction f2(2);  
    f2.Show();  
    Fraction f3(3, 4);  
    f3.Show();  
    // υπονοούμενη κλήση κατασκευαστή  
    f1 = 4;  
    f1.Show();  
    return 0;  
}
```

main.cpp

```
Fraction()  
0/1  
Fraction(int, int)  
2/1  
Fraction(int, int)  
3/4  
Fraction(int, int)  
4/1
```

# Καταστροφείς (destructors)

- Επιπρόσθετα στις ειδικές συναρτήσεις των κατασκευαστών, κάθε κλάση διαθέτει και μια ειδική συνάρτηση που ονομάζεται καταστροφέας (destructor).
- Ο καταστροφέας μοιάζει με τον προκαθορισμένο κατασκευαστή (δηλαδή τον κατασκευαστή χωρίς παραμέτρους) αλλά έχει το σύμβολο tilde ~ πριν το όνομά του.
- Οι καταστροφείς δεν μπορούν να έχουν παραμέτρους, άρα υπάρχει ένας μόνο καταστροφέας ανά κλάση.
  - Για παράδειγμα ο καταστροφέας της κλάσης `Fraction` θα είναι: `~Fraction()`;
- Όπως και με τους κατασκευαστές, οι καταστροφείς καλούνται αυτόματα (όχι ρητά).
- Οι καταστροφείς καλούνται αυτόματα ακριβώς πριν το αντικείμενο αποδεσμευτεί από το σύστημα, συνήθως όταν βγαίνει εκτός εμβέλειας (δηλαδή, όταν δεν είναι πλέον προσπελάσιμο από τον προγραμματιστή).
- Η τυπική εργασία ενός καταστροφέα είναι να πραγματοποιεί όποιες εργασίες αποδέσμευσης πόρων (συνήθως μνήμης) απαιτούνται, πριν το αντικείμενο αποδεσμευτεί.

# Παράδειγμα καταστροφεία

```
#include <iostream>
#include <string>
using std::string;
class Thing {
public:
    Thing(const char *n);
    ~Thing();
private:
    string name;
};
Thing::Thing(const char *n) {
    name = n;
    std::cout << "Constructor running for " << name << std::endl;
}
Thing::~~Thing() {
    std::cout << "Destructor running for " << name << std::endl;
}
void foo() {
    Thing Tfoo("Tfoo");
}
int main() {
    Thing T1("T1");
    std::cout << "Hello" << std::endl;
    foo();
    Thing T2("T2");
    return 0;
}
```

```
Constructor running for T1
Hello
Constructor running for Tfoo
Destructor running for Tfoo
Constructor running for T2
Destructor running for T2
Destructor running for T1
```



# Ερωτήσεις σύνοψης

- Ποιος είναι ο ρόλος της δεσμευμένης λέξη friend σε μια κλάση;
- Μια συνάρτηση που έχει γίνει friend σε μια κλάση είναι public ή private;
- Ποια είναι η διαφορά ανάμεσα στις δύο ακόλουθες κλήσεις:
  - `f3 = f1.Add(f2);`
  - `f3 = Add(f1,f2);`
- Τι είναι ο κατασκευαστής μετατροπής;
- Πώς ακυρώνουμε τις αυτόματες μετατροπές που γίνονται από έναν κατασκευαστή μετατροπής;

# Αναφορές

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