

**Ανοικτά Ακαδημαϊκά Μαθήματα**

**Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας**

**Αντικειμενοστρεφής Προγραμματισμός (Ε)**

**Ενότητα 4: Υπερφόρτωση τελεστών**

Κλειώ Σγουροπούλου

Τμήμα Μηχανικών Πληροφορικής ΤΕ

|  |  |
| --- | --- |
| Το περιεχόμενο του μαθήματος διατίθεται με άδεια Creative Commons εκτός και αν αναφέρεται διαφορετικά | Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους. |

Πίνακας Περιεχομένων

[Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 4 3](#_Toc402941589)

[Ο δείκτης this. 3](#_Toc402941590)

[Υπερφόρτωση τελεστών 4](#_Toc402941591)

[Υπερφόρτωση μοναδιαίων τελεστών 5](#_Toc402941592)

[Υπερφόρτωση δυαδικών τελεστών 6](#_Toc402941593)

[Υπερφόρτωση συντελεστών ανάθεσης 7](#_Toc402941594)

[Vectors 8](#_Toc402941595)

# Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 4

## Ο δείκτης this.

Οι συναρτήσεις-μέλη (member functions) μιας κλάσης έχουν αυτόματη πρόσβαση σε ένα δείκτη που ονομάζεται *this*, και δείχνει στο ίδιο το αντικείμενο. Με αυτό τον τρόπο, κάθε συνάρτηση αποκτά άμεση πρόσβαση στο ίδιο το αντικείμενο που την καλεί (π.χ. μπορεί να διαβάσει την διεύθυνσή του).

Μέσω του δείκτη this μια συνάρτηση μέλος ενός αντικειμένου αποκτά πρόσβαση στις μεταβλητές του αντικειμένου αυτού (κάτι που όμως μπορούσαμε να κάνουμε ούτως ει άλλως και χωρίς τον this). Μια πιο πρακτική χρήση του this είναι για επιτρέψει σε μια συνάρτηση μέλος να επιστρέψει έναν δείκτη στο ίδιο το αντικείμενο.

Έτσι, αν x είναι μια μεταβλητή σε μια κλάση, μια συνάρτηση μέλος της κλάσης μπορεί ισοδύναμα να χρησιμοποιήσει τις παρακάτω εντολές

void doSomething(){

x=21;

this->x=21; //ίδιο αποτέλεσμα

}

Στο πρόγραμμα που ακολουθεί η συνάρτηση getThis(), μέλος της κλάσης Action, ορίζεται να επιστρέφει την τιμή του δείκτη this.

#include <iostream> using namespace std;

class Action

{

int i;

public:

Action(int k);

int get\_member(void); Action \*getThis(void);

};

Action::Action(int k)

{

i=k;

}

int Action::get\_member(void)

{

return this->i; //μπορούσαμε να κάνουμε απλά: return i;

}

Action \*Action::getThis(void)

{

return this;

}

int main()

{

Action a(21);

cout<<"i= "<< a.get\_member()<<endl;

cout<<"\nThe address of the This pointer is:"<<a.getThis(); cout<<"\nThe address of the reference is:"<<&a;

return 0;

}

Πιο πολλά για τον δείκτη this θα δούμε παρακάτω.

## Υπερφόρτωση τελεστών

H υπερφόρτωση τελεστή αποτελεί μια ειδική περίπτωση υπερφόρτωσης συνάρτησης. Η C++ επιτρέπει στον προγραμματιστή να εκχωρεί πρόσθετες λειτουργίες στους βασικούς τελεστές της C++, όπως είναι αυτοί της πρόσθεσης, της αφαίρεσης κλπ. Με αυτό τον τρόπο, οι βασικοί τελεστές της C++ μπορούν, συσχετιζόμενοι με μια κλάση (και όχι μόνο), να έχουν ιδιαίτερο αποτέλεσμα όταν εφαρμόζονται σε αντικείμενα της κλάσης αυτής.

Για παράδειγμα, σε μια κλάση Matrix η οποία περιέχει ως member variable ένα πίνακα m[], μπορούμε να ορίσουμε ότι ο τελεστής +, όταν εφαρμόζεται σε αντικείμενα τύπου Matrix να εκτελεί την πρόσθεση πινάκων. Προφανώς αν ο τελεστής + εφαρμοστεί σε απλές μεταβλητές (όπως int, float κλπ.) θα εκτελεί τη γνωστή πράξη της πρόσθεσης.

Για να επιτευχθεί η υπερφόρτωση τελεστών πρέπει να οριστεί μια συνάρτηση μέλος ή μια συνάρτηση φιλική προς την κλάση η οποία θα υλοποιεί όλες τις επιμέρους λειτουργίες που θα γίνονται όταν χρησιμοποιείται ο υπερφορτωμένος τελεστής. Στην περίπτωση των πινάκων που προαναφέρθηκαν, η συνάρτηση του τελεστή θα αποκτά πρόσβαση στους πίνακες m[] των αντικειμένων Matrix που θα προστεθούν, και προσθέτει ένα προς ένα τα αντίστοιχα μέλη των πινάκων αυτών.

Η γενική μορφή μιας τέτοιας συνάρτησης είναι :

* τύπος όνομα\_κλάσης::operator#(λίστα\_ορισμάτων) όπου :
* **τύπος** : Ο τύπος επιστροφής του στοιχείου το οποίο θα είναι το αποτέλεσμα της
* πράξης.
* **όνομα\_ κλάσης** : Δηλώνει το όνομα της κλάσης με την οποία συσχετίζεται ο τελεστής
* **#** : Αποτελεί το τελεστή που μπορεί να υπερφορτωθεί. Μπορεί να είναι κάποιος από τους :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ! | ~ | + | - | \* | & | / | % |
| << | >> | < | <= | > | >= | == | != |
| ^ | | | && | > | += | -= | \*= | /= |
| ^= | |= | &= | %= | <<= | >>= | , | →\* |
| → | () | {} | = | ++ | -- | new | delete |

Σημειώνεται ότι εκτός από τον τελεστή = , η υπερφόρτωση τελεστών μπορεί να μεταβιβαστεί (inherited) και στις παραγόμενες κλάσεις της τάξης για την οποία ορίστηκε.

Προσοχή: Κάποιοι τελεστές δεν μπορούν να υπερφορτωθούν:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .Direct member selection | ? :conditional | ::Scope resolution | .\*Direct member selection | sizeof |

Είναι γεγονός ότι μπορούμε να ορίσουμε ένα τελεστή να εκτελεί μια πράξη πολύ διαφορετική από αυτή που εκτελείται όταν τον εφαρμόζουμε σε απλές μεταβλητές. Για παράδειγμα, μπορούμε να ορίσουμε τον τελεστή ‘+’ να εκτελεί αφαίρεση πινάκων. Όμως η υπερφόρτωση τελεστών για πράξεις άσχετες με τη βασική χρήση του τελεστή δεν αποτελεί καλή προγραμματιστική τεχνική γιατί μπορεί να μπερδέψει αυτούς που ενδεχομένως να χρησιμοποιήσουν την κλάση αυτή μελλοντικά.

## Υπερφόρτωση μοναδιαίων τελεστών

Με τον όρο μοναδιαίοι (unary) τελεστές εννοούμε αυτούς που εφαρμόζονται σε μία μόνο μεταβλητή, π.χ. οι i++, ++i, κ.α. Αντίστοιχα, δυαδικοί (binary) είναι οι τελεστές που εφαρμόζονται σε δύο μεταβλητές, π.χ. i+j, x\*y, κλπ.

#include <iostream> using namespace std; class Counter

{

private:

unsigned int count; //count

public:

Counter() : count(0) //constructor no args

{ }

Counter(int c) : count(c) //constructor, one arg

{ }

unsigned int get\_count() //return count

{ return count; }

Counter operator ++ () //increment count (prefix)

{ //increment count, then return

return Counter(++count); //an unnamed temporary object

} //initialized to this count

Counter operator ++ (int) //increment count (postfix)

{ //return an unnamed temporary

return Counter(count++); //object initialized to this

} //count, then increment count

};

int main()

{

Counter c1, c2; //c1=0, c2=0

cout << "\nc1=" << c1.get\_count(); //display cout << "\nc2=" << c2.get\_count();

++c1; //c1=1

c2 = ++c1; //c1=2, c2=2 (prefix)

cout << "\nc1=" << c1.get\_count(); //display cout << "\nc2=" << c2.get\_count();

c2 = c1++; //c1=3, c2=2 (postfix) cout << "\nc1=" << c1.get\_count(); //display again

cout << "\nc2=" << c2.get\_count() << endl;

return 0;

}

##### Άσκηση 1

Να υλοποιήσετε την κλάση Fraction η οποία αναπαριστά ένα κλάσμα της μορφής a/b

(αριθμητής/παρονομαστής). Μια τέτοια κλάση μπορεί να χρησιμοποιηθεί σε περιπτώσεις που δεν θέλουμε να χρησιμοποιήσουμε δεκαδικούς αριθμούς, αλλά κλάσματα.

Να υλοποιήσετε επίσης τους τελεστές ++ (prefix και postfix) ώστε να αυξάνουν το κλάσμα κατά ένα.

Να υλοποιήσετε και τους απαραίτητους κατασκευαστές, καθώς και μια συνάρτηση που εκτυπώνει το κλάσμα με την μορφή: a/b

## Υπερφόρτωση δυαδικών τελεστών

#include <iostream>

using namespace std;

class Matrix

{

private:

int arr[3];

public:

Matrix(int dim1, int dim2, int dim3)

{

arr[0] = dim1; arr[1] = dim2; arr[2] = dim3;

}

Matrix()

{

arr[0] = 0;

arr[1] = 0;

arr[2] = 0;

}

void print()

{

cout << "Matrix=("<<arr[0]<<", "<<arr[1]<<", "<<arr[2] <<")"<< endl;

}

Matrix operator+(Matrix m)

{

Matrix temp;

temp.arr[0] = arr[0] + m.arr[0];

temp.arr[1] = arr[1] + m.arr[1];

temp.arr[2] = arr[2] + m.arr[2];

return temp;

}

Matrix operator-(Matrix m)

{

Matrix temp;

temp.arr[0] = arr[0] - m.arr[0];

temp.arr[1] = arr[1] - m.arr[1];

temp.arr[2] = arr[2] - m.arr[2];

return temp;

}

};

int main()

{

Matrix m1(1,3,5);

Matrix m2(6,1,4);

Matrix m3;

m3 = m1 + m2;

m3.print();

system("pause");

return 0;

}

##### Άσκηση 2

Στην παραπάνω κλάση Fraction να υλοποιήσετε και τους τελεστές + και – για την πρόσθεση και αφαίρεση μεταξύ κλασμάτων.

Παράδειγμα:

2/3 + 4/5 22/15

Σημείωση: Μην επιχειρήσετε να κάνετε απλοποίηση του κλάσματος. Χρησιμοποιείστε το ελάχιστο κοινό πολλαπλάσιο.

## Υπερφόρτωση συντελεστών ανάθεσης

Στις προηγούμενες περιπτώσεις υπερφόρτωσης συντελεστών, η συνάρτηση υπερφόρτωσης επέστρεψε ένα νέο αντικείμενο, π.χ με την εντολή:

return Counter(++count);

Το νέο αυτό αντικείμενο μπορεί να ανατεθεί σε μια νέα μεταβλητή, πχ.:

c2 = ++c1;

Στην περίπτωση της υπερφόρτωσης των συντελεστών ανάθεσης, γενικώς δεν είναι καλό να δημιουργούνται τέτοια επιπλέον αντικείμενα. Για τον λόγο αυτό επιστρέφουμε το reference του ίδιου του αντικειμένου, αντί για ένα νέο αντίγραφό του. Για να το πετύχουμε αυτό χρησιμοποιούμε τον δείκτη this:

#include <iostream> using namespace std;

class alpha

{

private:

int data;

public:

alpha() //no-arg constructor

{ }

alpha(int d) //one-arg constructor

{ data = d; }

void display() //display data

{ cout << data; }

alpha& operator = (alpha& a) //overloaded = operator

{

data = a.data; //not done automatically

cout << "\nAssignment operator 1 invoked";

return \*this; //return copy of this alpha

}

alpha& operator = (int number) //overloaded = operator

{

data = number; //not done automatically

cout << "\nAssignment operator 2 invoked";

return \*this; //return copy of this alpha

}

};

int main()

{

alpha a1(37);

alpha a2, a3, a4;

a3 = a2 = a1; //invoke overloaded =, twice a4 = 2;

cout << "\na2="; a2.display(); //display a2 cout << "\na3="; a3.display(); //display a3 cout << "\na4="; a4.display(); //display a3 cout << endl;

return 0;

}

## Vectors

Οι Vectors θα παρουσιαστούν εκτενώς σε επόμενο μάθημα. Όμως, επειδή αποτελούν πολύ χρήσιμο εργαλείο για την αποθήκευση δεδομένων, καλό είναι να εξοικειωνόμαστε με την χρήση τους.

Στο παρακάτω παράδειγμα αρχικά δημιουργούμε ένα Vector ο οποίος θα περιέχει integers και στην συνέχεια εισάγουμε τιμές σε αυτόν. Γενικά η δήλωση

vector<MyClass> v;

δημιουργεί έναν Vector με όνομα v που έχει την δυνατότητα να περιέχει αντικείμενα της κλάσης MyClass.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v; //create a vector of integers

v.push\_back(10); //put values at end of array v.push\_back(11);

v.push\_back(12); v.push\_back(13);

v[0] = 20; //replace with new values v[3] = 23;

for(int j=0; j<v.size(); j++) //display vector contents cout << v[j] << ‘ ‘; //20 11 12 23

cout << endl; return 0;

}

Ουσιαστικά μοιάζει με ένα πίνακα στον οποίο όμως δεν χρειάζεται να ελέγξουμε τα όρια του. Ένα άλλο πλεονέκτημα είναι η εισαγωγή δεδομένων σε οποιοδήποτε μέρος του Vector:

#include <iostream>

#include <vector>

using namespace std;

int main()

{

int arr[] = { 100, 110, 120, 130 }; //an array of integers

vector<int> v(arr, arr+4); //initialize vector to array cout << “\nBefore insertion: ”;

for(int j=0; j<v.size(); j++) //display all elements

cout << v[j] << ‘ ‘;

v.insert( v.begin()+2, 115); //insert 115 at element 2 cout << “\nAfter insertion: ”;

for(j=0; j<v.size(); j++) //display all elements

cout << v[j] << ‘ ‘;

v.erase( v.begin()+2 ); //erase element 2

cout << “\nAfter erasure: ”;

for(j=0; j<v.size(); j++) //display all elements

cout << v[j] << ‘ ‘; cout << endl;

return 0;

}

Για περισσότερες πληροφορίες μπορείτε να ανατρέξετε στο <http://www.cplusplus.com/reference/stl/vector/>όπου θα βρείτε όλες τις συναρτήσεις που παρέχει η κλάση αυτή.

Παρακάτω φαίνονται οι συναρτήσεις τις οποίες υποστηρίζει ένας Vector:

|  |  |
| --- | --- |
| Vector constructors | create vectors and initialize them with some data |
| Vector operators | compare, assign, and access elements of a vector |
| assign | assign elements to a vector |
| at | returns an element at a specific location |
| back | returns a reference to last element of a vector |
| begin | returns an iterator to the beginning of the vector |
| capacity | returns the number of elements that the vector can hold |
| clear | removes all elements from the vector |
| empty | true if the vector has no elements |
| end | returns an iterator just past the last element of a vector |
| erase | removes elements from a vector |
| front | returns a reference to the first element of a vector |
| insert | inserts elements into the vector |
| max\_size | returns the maximum number of elements that the vector can hold |
| pop\_back | removes the last element of a vector |
| push\_back | add an element to the end of the vector |
| rbegin | returns a reverse\_iterator to the end of the vector |
| rend | returns a reverse\_iterator to the beginning of the vector |
| reserve | sets the minimum capacity of the vector |
| resize | change the size of the vector |
| size | returns the number of items in the vector |
| swap | swap the contents of this vector with another |

(Από το site: [http://www.cppreference.com/cppvector/index.html)](http://www.cppreference.com/cppvector/index.html%29)

**Άλλα παραδείγματα:**

Εισαγωγή αριθμών στο τέλος του vector

vector<**int**> the\_vector;

for( int i = 0; i < 10; i++ )

{

the\_vector.push\_back( i );

}

for(i = 0; i < 10; i++ )

{

cout << the\_vector.pop\_back();

}

Αποθήκευση strings στον vector και ταξινόμηση

vector<string> words; string str;

while( cin >> str ) words.push\_back(str);

sort( words.begin(), words.end() );

cout << "In alphabetical order, the first word is '" << words.front() << "'." << endl;

Αποθήκευση **αντικειμένων τύπου Vehicle** στον vector

#include <iostream>

#include <string>

#include <vector>

using namespace std; class Vehicle

{

public:

string model;

};

int main()

{

Vehicle v;

vector<Vehicle> vec; v.model = "Yaris";

vec.push\_back(v); v.model = "316";

vec.push\_back(v);

for(int i=0; i<vec.size(); i++)

cout << "Model: " << vec[i].model << endl;

system("pause");

return 0;

}

##### Άσκηση 3

Σε συνέχεια των Ασκήσεων 1 και 2, να υλοποιήσετε μια συνάρτηση main και μέσα σε αυτήν να ορίσετε ένα

Vector ο οποίος θα αποθηκεύει αντικείμενα τύπου Fraction.

Στην συνέχεια να δημιουργήσετε μερικά αντικείμενα Fraction και να τα αποθηκεύσετε σε αυτό τον Vector.

Στο τέλος να εκτυπώσετε ένα προς ένα όλα τα αντικείμενα που είναι αποθηκευμένα σε αυτόν, διατρέχοντας τον vector και χρησιμοποιώντας την συνάρτηση εκτύπωσης κλασμάτων της Άσκησης 1.

|  |
| --- |
| **Ανοικτά Ακαδημαϊκά Μαθήματα****Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας** |
| **Τέλος Ενότητας** |
| **Χρηματοδότηση*** Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Αθήνας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

 |

**Σημειώματα**

**Σημείωμα Αναφοράς**

Copyright ΤΕΙ Αθήνας, Κλειώ Σγουροπούλου, 2014. Κλειώ Σγουροπούλου. «Αντικειμενοστρεφής Προγραμματισμός». Ενότητα 4: «Υπερφόρτωση τελεστών». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: [ocp.teiath.gr](https://ocp.teiath.gr/).

**Σημείωμα Αδειοδότησης**

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

* που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
* που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
* που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

**Διατήρηση Σημειωμάτων**

* Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
* Το Σημείωμα Αναφοράς
* Το Σημείωμα Αδειοδότησης
* Τη δήλωση Διατήρησης Σημειωμάτων
* Το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους.