ΥΠΕΡΦΟΡΤΩΣΗ ΤΕΛΕΣΤΩΝ

Τι είναι υπερφόρτωση τελεστών

Η ιδιότητα με την οποία στην C++ μπορούμε να έχουμε τελεστές που να ενεργούν διαφορετικά μέσα στο ίδιο πρόγραμμα την ονομάζουμε υπερφόρτωση.

Το χαρακτηριστικό αυτό της C++ δηλαδή την διαφορετική χρησιμοποίηση των τελεστών μέσα στο ίδιο πρόγραμμα το λέμε πολυμορφισμό.

Άρα μία γλώσσα προγραμματισμού είναι αντικειμενοστραφής όταν συνδυάζει τα χαρακτηριστικά των κλάσεων-αντικειμένων, της κληρονομικότητας και του πολυμορφισμού.

Γιατί χρειαζόμαστε την υπερφόρτωση τελεστών

Οι τελεστές π.χ. +, -, *, /, <<, ==, <, >, != κ.τ.λ. όπως τους ξέρουμε ενεργούν μόνο επάνω σε μεταβλητές. Π.χ:

```
z = x + y   ή   a += (b * c)   ή   i < j
```

Τελεστές με αντικείμενα

Τι γίνεται όμως όταν έχουμε αντικείμενα;

Μπορούμε δηλαδή να αυξήσουμε την τιμή ενός αντικειμένου;

```
(++object)
```

ή

Να κάνουμε πράξεις με τις τιμές αντικειμένων;

```
(object3 = object1 + object2)
```

εδώ ο compiler θα εντοπίσει λάθος. Εδώ θα χρειαστούμε την ιδιότητα της υπερφόρτωσης τελεστών.

Για να το κάνουμε αυτό πρέπει να χρησιμοποιήσουμε στην συνάρτηση που θα χρειαστεί να αυξήσει τιμή στο αντικείμενο, την δεσμευμένη από τη γλώσσα C++ λέξη κλειδί **operator**.
Η λέξη αυτή επεκτείνει την χρήση του εκάστοτε τελεστή (++, --, +, - κλπ) από τις μεταβλητές και στα αντικείμενα.
Με την λέξη operator φτιάχνουμε συνάρτηση αύξησης ή μείωσης της τιμής.

Υπάρχουν δύο μορφές χρήσης του operator,

- η **προθεματική** όπου ο τελεστής βρίσκεται αριστερά από το δεδομένο-μέλος και
- η **επιθεματική** όπου ο τελεστής βρίσκεται δεξιά από το δεδομένο- μέλος.

Τις δύο μορφές τις αναλύουμε παρακάτω.

Προθεματική χρήση του operator

Εάν έχουμε προθεματική μορφή τελεστή (++i) τότε ο operator γίνεται:

```
void  operator ++ (){
   ++i;
}
```

Τώρα στο αντικείμενο που θα φτιάξουμε θα εφαρμόσουμε κατευθείανεπάνω του τον τελεστή ++ χωρίς να υπάρξει λάθος του compiler δηλ.

```
++object
```

Επιθεματική χρήση του operator

Εάν έχουμε επιθεματική μορφή τελεστή (i++) τότε ο operator γίνεται:

```
void  operator ++ (int){
   i++;
}
```

Το int μέσα στην παρένθεση δεν σημαίνει ότι υπάρχει ακέραιος αριθμός, αλλά κάνει τον compiler να ξεχωρίσει πως θα χρησιμοποιήσει τον operator.

Μπορούμε τώρα να εφαρμόσουμε τον τελεστή ++ πάνω στο αντικείμενο δηλ

```
object++
```

Υπερφόρτωση του τύπου x+=y

Κάπως έτσι μπορούμε να χρησιμοποιήσουμε υπερφόρτωση του τελεστή ανάθεσης τιμής με την χρήση του operator ως εξής:

```
void  operator ++(){
   i += 2;
}
```

Υπερφόρτωση του τελεστή άθροισης (+)

Για να μπορέσουμε να προσθέσουμε αντικείμενα δηλ.

```
Acc3 = Acc1 + Acc2;
```

θα πρέπει να χρησιμοποιήσουμε τον πιο κάτω κώδικα:

```
Account operator + (Account anAcc){
    Account temp;
    temp.balance = balance + anAcc.balance;
    return temp;
}
```

Τώρα μπορούμε να προχωρήσουμε στην αρχική έκφραση και να προσθέσουμε τις τιμές των αντικειμένων σαν να προσθέτουμε απλές μεταβλητές. Η συνάρτηση έχει την τιμή του `Acc1` και το όρισμα μέσα στην παρένθεση είναι το `Acc2`.

Υπερφόρτωση συγκριτικού τελεστή >

Μπορούμε με το ίδιο σκεπτικό να υπερφορτώσουμε και συγκριτικούς τελεστές. Η πρακτική αυτή μας βοηθά να συγκρίνουμε απευθείας τιμές αντικειμένων μέσα στην `main`() δηλ.

```
Acc1 > Acc2
```

θα πρέπει να χρησιμοποιήσουμε τον πιο κάτω κώδικα:

```
bool  operator > (Account anAcc){
    if(balance > anAcc.balance)
       return true;
    else
       return false;
}
```

Εδώ η μέθοδος επιστρέφει **`true`** αν το υπόλοιπο του αντικειμένου που την κάλεσε είναι μεγαλύτερο από το υπόλοιπο του αντικειμένου που έρχεται ως όρισμα της μεθόδου (αντικείμενο **anACC**).

Ποιοι τελεστές δεν υπερφορτώνονται

Όλοι οι τελεστές μπορούν να υπερφορτωθούν εκτός από τους παρακάτω:

Τελεστής	*Περιγραφή*
.	Τελεστής κλήσης (τελεία)
.*	Τελεστής δείκτη σε μέλος
::	Τελεστής διακρίβωσης εμβέλειας
?:	Τριαδικός τελεστής υπό όρους.

ΠΑΡΑΔΕΙΓΜΑΤΑ ΥΠΕΡΦΟΡΤΩΣΗΣ ΤΕΛΕΣΤΩΝ

Παράδειγμα 1. Υπερφόρτωση του τελεστή αύξησης κατά 1 (++)

```
#include <iostream>
using namespace std;
class metritis{
private:
	int met;
public:
	 metritis(){  //constructor
		met=0;
	}
	void calc_met(){ //μέθοδος για αύξηση του δεδομένου met κατά ένα
		++met;
	}
	int  get_met(){ //μέθοδος για επιστροφή τιμής του δεδομένου met
		 return met;
	}
};
int main(){
	metritis met1, met2; //2 αντικείμενα με αρχική τιμή 0
	int m, k, i = 0; //τοπικές μεταβλητές

	++i; //αύξηση της μεταβλητής i κατά 1
	++i; //αύξηση της μεταβλητής i κατά 1
	cout << "\nmetriths i=" << i;  //εκτύπωση της τιμής της μεταβλητής i
	cout << "\n\n";

	m = met1.get_met(); //επιστροφή τιμής του met1 και απόδοση στην μεταβλητή m (m = 0)
	k = met2.get_met(); //επιστροφή τιμής του met2 και απόδοση στην μεταβλητή k (k = 0)
	cout << "\nmetriths met1= "<< m; //εκτύπωση της τιμής της μεταβλητής m
	cout << "\nmetriths met2= "<< k; //εκτύπωση της τιμής της μεταβλητής k
	cout << endl;

	met1.calc_met(); //αύξηση κατά 1 του αντικειμένου met1 (met1 = 1)
	met2.calc_met(); //αύξηση κατά 1 του αντικειμένου met2 (met2 = 1)
	met1.calc_met(); //αύξηση κατά 1 του αντικειμένου met1 (met1 = 2)
	met1.calc_met(); //αύξηση κατά 1 του αντικειμένου met1 (met1 = 3)

	 //εκτύπωση της τιμής του αντικειμένου met1
	 cout << "\netriths met1=" << met1.get_met();

	 //εκτύπωση της τιμής του αντικειμένου met2
	cout << "\netriths met2=" << met2.get_met();

	 /*πρόσθεση των νέων τιμών των αντικειμένων
	 και απόδοση του αποτελέσματος στην μεταβλητή k */
	k = met1.get_met() + met2.get_met();

	 //εκτύπωση της νέας τιμής της μεταβλητής k
	cout << "\nto athroisma einai :" << k << endl;
}
```

Εάν γράψουμε ++met1 προσπαθώντας να αυξήσουμε την τιμή του αντικειμένου ο compiler θα μας δώσει λάθος. Γι αυτόν τον λόγο χρησιμοποιούμε την λέξη κλειδί operator με τον τελεστή ++ αντί της συνάρτησης calc_met(). Τώρα εάν χρησιμοποιήσουμε την έκφραση **++met1** ο compiler θα αυξήσει την τιμή του αντικειμένου χωρίς λάθος. Αυτό φαίνεται στο πιο κάτω πρόγραμμα.

```
#include <iostream>
……
	void operator ++(){ //υπερφόρτωση τελεστή ++
		++met;
	}
	int get_met(){ //μέθοδος για την επιστροφή της τιμής του met
		return met;
	}
};
int main(){
……
……

	cout << "\nmetriths met2= " << k;
	cout << endl;

	++met1; //αύξηση τιμής του αντικειμένου met1
	++met2; //αύξηση τιμής του αντικειμένου met2
	++met1; //αύξηση τιμής του αντικειμένου met1
	++met1; //αύξηση τιμής του αντικειμένου met1
……
……
	cout << "\nto athroisma einai :" << k << endl; //εκτύπωση του k
}
```

Στο πιο πάνω παράδειγμα έχουμε προθεματική χρήση του operator.

Για επιθεματική χρήση του operator η αλλαγή είναι πιο κάτω.

```
#include <iostream>
……
	//επιθεματική υπερφόρτωση του τελεστή ++
	 void operator ++(int){
		met++;
	}
	int get_met(){ //μέθοδος για την επιστροφή της τιμής του met
		return met;
	}
};
void main(){
……
……
	cout << "\nmetriths met2= " << k;
	cout << endl;
	met1++; //αύξηση τιμής του αντικειμένου met1
	met2++; //αύξηση τιμής του αντικειμένου met2
	met1++; //αύξηση τιμής του αντικειμένου met1
	met1++; //αύξηση τιμής του αντικειμένου met1
……
……

	cout << "\nto athroisma einai :" << k << endl; //εκτύπωση του k
}
```

Παράδειγμα 2.Προθεματικη και επιθεματική υπερφόρτωση

```
#include <iostream>
using namespace std;
class test {
private:
	unsigned int met;
public:
	test(){ //constructor χωρίς όρισμα
		met = 0;
	}
	test(int x) { //constructor με όρισμα
		met=x ;
	}
	test operator ++(){ //προθεματική υπερφόρτωση
		return test(++met);
	}
	test operator ++(int){ //επιθεματική υπερφόρτωση
		return test(met++);
	}
	int get_met(){  //μέθοδος για την επιστροφή της τιμής του met
		return met;
	}
};
int main(){
	test t1, t2;
	cout << "\nt1 = " << t1.get_met(); //(t1 = 0)
	cout << "\nt2 = " << t2.get_met(); //(t2 = 0)

	++t1; //προθεματική υπερφόρτωση   (t1 = 1)
	++t1; //προθεματική υπερφόρτωση   (t1 = 2)

	t2 = ++t1; //(t1 = 3, t2 = 3)
	cout << "\nt1 = " << t1.get_met(); //(t1 = 3)
	cout << "\nt2 = " << t2.get_met(); //(t2 = 3)

	t2 = t1++; //επιθεματική υπερφόρτωση (t2 = 3, t1 = 4)
	t2 = t1++; //επιθεματική υπερφόρτωση (t2 = 4, t1 = 5)

	cout << "\nt1 = " << t1.get_met(); //(t1 = 5)
	cout << "\nt2 = " << t2.get_met(); //(t2 = 4)

	cout << "\n\n";
}
```

Έξοδος: *1^{Ο} cout:* ***t1=0, t2=0*** | *2^{Ο} cout:* ***t1=3, t2=3*** | *3^{Ο} cout:* ***t1=5, t2=4***

Παράδειγμα 3.Υπερφορτωση αριθμητικού τελεστή άθροισης (+)

```
#include <iostream>
using namespace std;

class Accounts{
private:
	float balance;
public:
	Accounts(){
		balance = 0;
	}
	~ Accounts(){}
```

```
	//Ορισμός συνάρτησης για υπερφόρτωση του τελεστή άθροισης (+)
	Accounts operator + (Accounts anAccount){
		Accounts sum;
		sum.balance = balance + anAccount.balance;
		return sum;
	}
	void withdrawal(float amount){
		if (amount <= balance)
		balance = balance - amount;
		else{
			cout << "Warning!!!!!!" the withdrawal amount << endl;
			cout << " is greater than the balance of the account" << endl;
		}
	}
	void deposit(float amount){
		balance += amount;
	}
	float getBalance(){
		return balance;
	}
};
int main(){
	Accounts Acc1, Acc2, Acc3; //Δημιουργία αντικειμένων
	cout << endl;
	cout << "START OF TRANSACTIONS" << endl;

	cout << "========================================" << endl;
	cout << " INITIAL BALANCES" << endl; //Αρχικά υπόλοιπα αντικειμένων
	cout << "Acc1: "<< Acc1.getBalance() << " Euro" << endl;
	cout << "Acc2: "<< Acc2.getBalance() << " Euro" << endl;
	cout << "Acc3: "<< Acc3.getBalance() << " Euro" << endl;
	cout << "========================================" << endl;

	cout << " DEPOSIT OF 100 EUROS FROM THE OBJECT Acc1" << endl;
	Acc1.deposit(100.0); //Κατάθεση από το αντικείμενο Acc1
	cout << " DEPOSIT OF 100 EUROS FROM THE OBJECT Acc2" << endl;
	Acc2.deposit(200.0); //Κατάθεση από το αντικείμενο Acc2

	cout << "========================================" << endl;
	cout << " THE BANK BALANCES ARE NOW…… " << endl; //Νέα υπόλοιπα αντικειμένων
	cout << "Acc1: " << Acc1.getBalance() << " Euro" <<endl;
	cout << "Acc2: " << Acc2.getBalance() << " Euro" << endl;
	cout << "Acc3: " << Acc3.getBalance() << " Euro" << endl;
	cout << "========================================" << endl;

	cout << " WITHDRAW 100 EUROS FROM THE ITEM Acc1" << endl;
	Acc1. withdrawal (70.0); //Ανάληψη από το αντικείμενο Acc1

	cout << "========================================" << endl;
	cout << " THE BANK BALANCES ARE NOW…… " << endl;  //Νέα υπόλοιπα αντικειμένων
	cout << "Acc1: " << Acc1.getBalance() << " Euro" <<endl;
	cout << "Acc2: " << Acc2.getBalance() << " Euro" << endl;
	cout << "Acc3: " << Acc3.getBalance() << " Euro" << endl;
	cout << "========================================" << endl;

	cout << " ADDING THE BANK BALANCES OF Acc1 TO Acc2" << endl;
	cout << "========================================" << endl;
	Acc3 = Acc1 + Acc2; //Άθροιση υπολοίπων με υπερφόρτωση του τελεστή άθροισης

	cout << " THE BANK BALANCES ARE NOW…… " << endl; //Νέα υπόλοιπα αντικειμένων
	cout << "Acc1: " << Acc1.getBalance() << " Euro" <<endl;
	cout << "Acc2: " << Acc2.getBalance() << " Euro" << endl;
	cout << "Acc3: " << Acc3.getBalance() << " Euro" << endl;
```

```
        cout << "========================================" << endl;

        cout << "END OF TRANSACTIONS" << endl;
        cout << "========================================" << endl;
        cout << endl;
}
```