

# ΤΕΛΕΣΤΕΣ ΤΗΣ C++

- ❖ Οι τελεστές της C++ χωρίζονται σε δύο μεγάλες κατηγορίες:
  - i. **Μονήρεις (unary):** Οι τελεστές αφορούν ένα μόνο όρο
  - i. **Διαδικοί (binary):** Οι τελεστές αφορούν δύο όρους, τον αριστερό και το δεξιό όρο της έκφρασης
  - ii. **Τριαδικοί (ternary):** Οι τελεστές αφορούν τρεις όρους  
Στη πράξη μόνο ο υπό συνθήκη τελεστής

# ΤΕΛΕΣΤΕΣ ΣΥΜΠΛΗΡΩΜΑΤΙΚΟΤΗΤΑΣ

❖ Τρεις τελεστές μονήρεις που λειτουργούν από δεξιά προς αριστερά.

## ◊ Αρνητικό πρόσημο

- ◆ Συντακτικός τύπος:  $-a$
- ◆ Αποτέλεσμα: το  $a$  με αντίθετο πρόσημο

## ◊ Λογική άρνηση (NOT)

- ◆ Συντακτικός τύπος:  $!e$
- ◆ Αποτέλεσμα: λογικά αντίθετο από εκείνο της λογικής έκφρασης  $e$

## ◊ Ψηφιακό συμπλήρωμα

- ◆ Συντακτικός τύπος:  $\sim a$
- ◆ Αποτέλεσμα: μια δυαδική τιμή, όση του  $a$ , αφού γίνουν όλα τα μηδέν ένα και αντίστροφα

# ΠΡΟΤΕΡΑΙΟΤΗΤΕΣ ΤΕΛΕΣΤΩΝ

- ❖ Γενικά οι τελεστές ομαδοποιούνται από αριστερά
  - ◊ Εκτός από τους μοναδιαίους τελεστές και τους τελεστές αντικατάστασης που ομαδοποιούνται από δεξιά
$$a = b = c \rightarrow a = (b = c)$$
$$a + b + c \rightarrow (a + b) + c$$
$$*p++ \rightarrow *(p++)$$
- ❖ Υπάρχει καλά ορισμένη προτεραιότητα τελεστών, π.χ.
$$a + b * c \rightarrow a + (b * c)$$
- ❖ Παρόλα αυτά καλό είναι να χρησιμοποιούμε παρενθέσεις για να κάνουμε σαφή τη σειρά προτεραιότητας και για να αποφεύγουμε λάθη

# ΤΕΛΕΣΤΕΣ ΜΕΓΕΘΟΥΣ ΜΝΗΜΗΣ

- ❖ Ένας χρήσιμος τελεστής είναι ο: **sizeof**
- ❖ Επιστρέφει το **μέγεθος της μνήμης** που απαιτείται για αποθήκευση κάποιου τύπου δεδομένων (ή και σύνθετης μεταβλητής).
- ❖ Γενικός συντακτικός τύπος:  
**sizeof(όνομα τύπου δεδομένων ή σύνθετης μεταβλητής)**
- ❖ Παράδειγμα: Αν δοθεί η εντολή  
  
**f=sizeof(float) ;**  
  
η τιμή του **f** θα είναι 4 bytes για το 90% των μηχανών

# ΠΟΛΛΑΠΛΑΣΙΑΣΤΙΚΟΙ ΤΕΛΕΣΤΕΣ (1/2)

- ❖ Εφαρμόζονται πάντα σε δύο όρους με επίδραση από αριστερά προς τα δεξιά.
- ❖ Υπάρχουν τρεις πολλαπλασιαστικοί τελεστές με συντακτικούς τύπους:  
$$\mathbf{x * y} \ , \ \mathbf{x / y} \ , \ \mathbf{x \% y}$$
  - ◊ ο τελεστής  $*$  επιστρέφει την τιμή του γινομένου των δύο όρων
  - ◊ ο τελεστής  $/$  επιστρέφει το αποτέλεσμα της διαίρεσης του  $x$  δια του  $y$
  - ◊ ο τελεστής  $\%$  επιστρέφει το υπόλοιπο της διαίρεσης του  $x$  δια του  $y$
- ❖ Οι τελεστές  $*$  και  $/$  απαιτούν τα  $x$  και  $y$  να έχουν πραγματικές τιμές, ενώ ο  $\%$  λειτουργεί μεταξύ ακέραιων τιμών.

# ΠΟΛΛΑΠΛΑΣΙΑΣΤΙΚΟΙ ΤΕΛΕΣΤΕΣ (2/2)

- ❖ Εάν το  $x$ ,  $y$  δεν είναι του ίδιου τύπου μεταβλητές ή σταθερές ή εκφράσεις, ο τελεστής εκτελεί αυτόματα τις μετατροπές τύπων. Προσοχή, δεν ελέγχεται η περίπτωση υπερχείλισης τιμών (overflow ή underflow).
- ❖ Η επιστρεφόμενη πληροφορία χάνεται αν είναι ασυμβίβαστη με τους τύπους των όρων.
- ❖ Παράδειγμα:

```
int i=20, j=7, m;  
double x=3.0, y;  
y=x*i-1; //1  
m=i/j; //2  
m=i%j; //3
```

- ◇ Η σχέση 1 θα δώσει 59.0 της μορφής double
- ◇ Η σχέση 2 θα δώσει 2
- ◇ Η σχέση 3 θα δώσει την τιμή 6 που είναι και το υπόλοιπο της διαίρεσης μεταξύ των  $i$  και  $j$ .

# ΠΡΟΣΘΕΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

- ❖ Εφαρμόζονται σε δύο όρους με επίδραση από αριστερά στα δεξιά.
- ❖ Υπάρχουν δύο τελεστές με συντακτικούς τύπους:

$$x+y , x-y$$

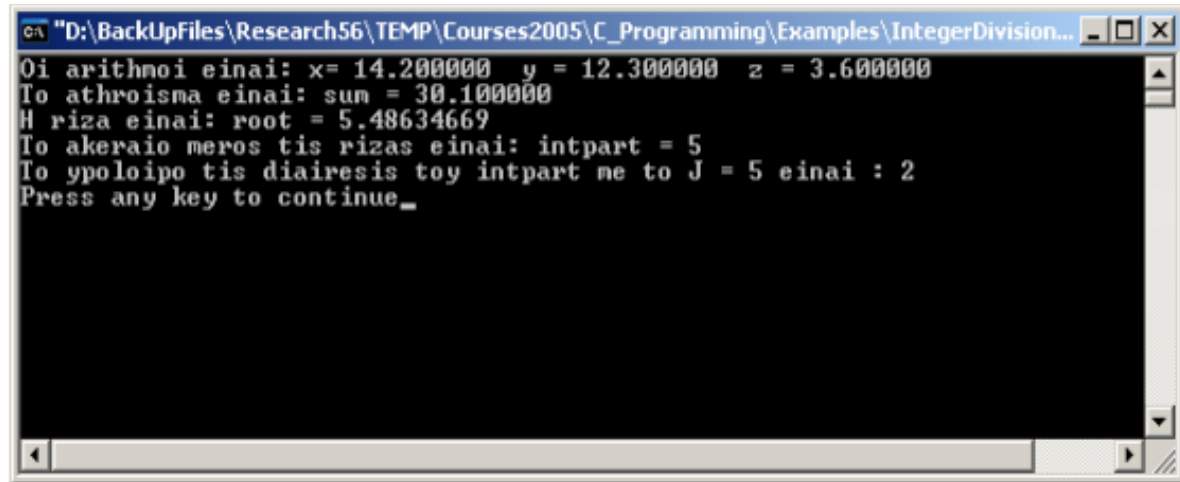
- ◊ ο τελεστής  $+$  επιστρέφει την τιμή της πρόσθεσης του  $x$  με τον  $y$
- ◊ ο τελεστής  $-$  επιστρέφει την τιμή της αφαίρεσης του  $y$  από τον  $x$

- ❖ Και στους δύο τελεστές οι όροι  $x$  και  $y$  πρέπει να είναι αριθμητικού τύπου.

Προσοχή, αν τα  $x$  και  $y$  δεν είναι του ίδιου τύπου, τότε ο προσθετικός τελεστής προβαίνει αυτόματα σε μετατροπές τύπων, χωρίς να ελέγχεται η περίπτωση υπερχείλισης τιμών.

# ΠΡΟΣΘΕΤΙΚΟΙ ΤΕΛΕΣΤΕΣ

```
#include <stdio.h>
#include <math.h>
void main()
{
    double x = 14.2, y = 12.3, z = 3.6;
    double sum, root;
    int intpart, J=3, remainder;
    sum= x + y + z;
    printf("Oi arithmoi einai: x= %lf y = %lf z = %lf \n", x, y, z);
    printf("To athroisma einai: sum= %lf\n", sum);
    root=pow(sum,0.5);
    printf("H riza einai: root = %10.8lf\n", root);
    intpart=(int)root;
    remainder = intpart % J;
    printf("To akeraio meros tis rizas einai: intpart = %d\n", intpart);
    printf("To ypoloipo tis diairesis toy intpart me to J = %d einai : %d\n", intpart,
    remainder);
    return;
}
```



```

D:\BackUpFiles\Research56\TEMP\Courses2005\C_Programming\Examples\IntegerDivision...
Oi arithmoi einai: x= 14.200000 y = 12.300000 z = 3.600000
To athroisma einai: sum = 30.100000
H riza einai: root = 5.48634669
To akeraio meros tis rizas einai: intpart = 5
To ypoloipo tis diairesis toy intpart me to J = 5 einai : 2
Press any key to continue_
```



# Overflow / underflow

- ❖ Το over/underflow συμβαίνει όταν στο πιο σημαντικό ψηφίο , το εισερχόμενο κρατούμενο (carry in), είναι διαφορετικό από το εξερχόμενο κρατούμενο (carry out).

# Overflow / underflow

Η πράξη  $-32 \leq 14 + 9 \leq 31$  σε 6 bits 2's complement είναι επιτρεπτή:

Τιμή θέσης	$-2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	Δεκαδική αναπαράσταση
	0	0	1	1	1	0	= 14
	0	0	1	0	0	↓	= 9
	0	1	0	1	1	1	= 23

**carry out = 0, carry in=0** ⇒ αποτέλεσμα σωστό.

Η πράξη  $25 + 18 > 31$  σε 6 bits 2's complement δεν είναι επιτρεπτή

Τιμή θέσης	$-2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	Δεκαδική αναπαράσταση
	0	1	1	0	0	1	= 25
	0	1	0	0	1	0	= 18
	1	0	1	0	1	1	= -21

**carry out = 0, carry in=1** ⇒ **overflow** ⇒ αποτέλεσμα λανθασμένο.

Η πράξη  $-32 \leq 17 - 13 \leq 31$  σε 6 bits 2's complement είναι επιτρεπτή:

Τιμή θέσης	$-2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	Δεκαδική αναπαράσταση
	0	1	0	0	0	1	= 17
	1	1	0	0	1	1	= -13
	0	0	0	1	0	0	= 4

**carry out = 1, carry in=1** ⇒ αποτέλεσμα σωστό.

Η πράξη  $-8 - 31 < -32$  σε 6 bits 2's complement δεν είναι επιτρεπτή

Τιμή θέσης	$-2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	Δεκαδική αναπαράσταση
	1	1	1	0	0	0	= -8
	1	0	0	0	0	1	= -31
	0	1	1	0	0	1	= 25

**carry out = 1, carry in=0** ⇒ **underflow** ⇒ αποτέλεσμα λανθασμένο.

# ΤΕΛΕΣΤΕΣ ΚΙΝΗΣΗΣ

- ❖ Οι τελεστές κίνησης ή ολίσθησης εφαρμόζονται σε δύο όρους με γενικούς συντακτικούς τύπους:

$$a \ll b, a \gg b$$

- ◊ Η επιστρεφόμενη τιμή του τελεστή  $\ll$  είναι το  $a$  αφού μετατοπισθεί η δυαδική του αντίστοιχη τιμή τόσα bits όση η τιμή του  $b$  **προς τα αριστερά**.
- ◊ Ο τελεστής  $\gg$  προκαλεί ανάλογη κίνηση τιμής του  $a$  **προς τα δεξιά**.
- ◊ Στην αριστερή μετακίνηση τα δεξιά bits που δημιουργούνται γεμίζονται με μηδέν.
- ◊ Στη δεξιά κίνηση
  - ◆ αν το  $a$  είναι unsigned τύπου, τότε τα αριστερά bits που δημιουργούνται γεμίζονται επίσης με μηδέν,
  - ◆ αν το  $a$  δεν είναι unsigned τύπου, τότε είναι άγνωστο τι θα προκύψει.

# Παράδειγμα

```
.....  
unsigned int i=2, j=2, z;  
z=i<<j;  
z = z >>3;  
printf ("z=%i", z);  
.....
```

- ❖ Οι μεταβλητές  $i$ ,  $j$  και  $z$  είναι απόλυτοι ακέραιοι.
- ❖ Η μεταβλητή  $i$  έχει αρχική τιμή 2, η οποία σε δυαδικό σύστημα ισοδυναμεί με την τιμή 0010.
- ❖ Στη συνέχεια, σαν τιμή του  $z$  τίθεται η τιμή του  $i$  αφού γίνει αριστερή μετατόπιση σε επίπεδο bits, κατά  $j$  θέσεις, δηλαδή κατά 2 θέσεις.
- ❖ Οπότε, θα προκύψει σαν τιμή του  $z$  ο δυαδικός αριθμός 1000, ο οποίος δεν είναι άλλος από τον αριθμό 8 του δεκαδικού συστήματος.
- ❖ Στη συνέχεια, τίθεται σαν τιμή του  $z$ , αυτή που υπάρχει μέχρι τώρα, αφού υποστεί δεξιά μετατόπιση σε επίπεδο bits κατά 3 θέσεις, οπότε θα προκύψει ο δυαδικός αριθμός 0001, ο οποίος βέβαια ισοδυναμεί με τον ακέραιο 1.
- ❖ Τελικά λοιπόν, θα γίνει εμφάνιση σαν τιμή του  $z$  στην οθόνη, το 1.

# ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (1/3)

- ❖ Ελέγχουν αν δύο όροι ταυτίζονται τις τιμές τους ή είναι άνισοι σε κάθε δυνατό τρόπο ανισότητας.
- ❖ Λειτουργούν από αριστερά προς δεξιά.
- ❖ Συντακτικοί τύποι:
  - a<b** : a μικρότερο του b
  - a>b** : a μεγαλύτερο του b
  - a<=b** : a μικρότερο ή και ίσο του b
  - a>=b** : a μεγαλύτερο ή και ίσο του b
  - a==b** : a ίσο του b
  - a!=b** : a διάφορο του b
- ◊ Το επιστρεφόμενο αποτέλεσμα είναι 1 αν ισχύει η σχέση, διαφορετικά μηδέν.
- ◊ Ο τύπος της επιστρεφόμενης τιμής είναι int.
- ◊ Οι όροι μπορούν να είναι ακέραιοι, πραγματικοί ή δείκτες. Αν ο ένας όρος είναι ακέραιος και ο άλλος πραγματικός, γίνονται αυτόματες μετατροπές και τελικά εξάγεται η ορθή επιστρεφόμενη τιμή.

## ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (2/3)

- ❖ Στους τελεστές  $==$  και  $!=$  μπορεί ο ένας όρος ή και οι δύο να είναι τύπου `enum`, οπότε ο μεταφραστής θεωρεί τις εσωτερικές ακέραιες τιμές των τύπων αυτών για να επιστρέψει επί.

# ΣΧΕΣΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (3/3)

- ❖ Προσοχή στη διάκριση μεταξύ του απλού τελεστή  $=$  και του σχεσιακού τελεστή  $==$ .
  - ◇  $O =$  απλά δηλώνει αντικατάσταση (εκχώρηση τιμής)
  - ◇  $O ==$  ελέγχει αν ισχύει ή όχι ταύτιση των τιμών των δύο όρων.
- ❖ Υπενθυμίζεται ότι είναι πολύ σπάνιο γεγονός η ταύτιση τιμών μεταξύ δύο float τύπων και επιτυγχάνεται μόνο εάν εφαρμοστεί κάποια προσέγγιση.

# ΨΗΦΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (1/3)

- ❖ Επιδρούν στις δυαδικές τιμές ακέραιων τύπων, από αριστερά προς δεξιά.
- ❖ Το επιστρεφόμενο αποτέλεσμα είναι δυαδικός αριθμός ο οποίος αντιστοιχεί σε κάποια ακέραια τιμή.
- ❖ Συντακτικοί τύποι:

**$a \& b$  ,  $a | b$  ,  $a ^ b$**

- ❖ Ο **&** ονομάζεται **ψηφιακό ΚΑΙ (bitwise AND)** και επιστρέφει μια ακέραια τιμή που αντιστοιχεί στο δυαδικό αριθμό που προέρχεται από την πράξη με τον προτασιακό τύπο:  
«Αν το bit  $i$  της δυαδικής τιμής του  $a$  είναι 1 και το bit  $i$  της δυαδικής τιμής του  $b$  είναι 1, τότε το bit  $i$  του αποτελέσματος είναι 1, διαφορετικά είναι 0. Η εφαρμογή της πρότασης για κάθε  $i$  δίνει τον τελικό επιστρεφόμενο αριθμό.»



# ΨΗΦΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (2/3)

- ❖ Ο  $|$  ονομάζεται **ψηφιακό εγκλειστικό Η (bitwise inclusive OR)** και επιστρέφει μια ακέραια τιμή που αντιστοιχεί στο δυαδικό αριθμό που προέρχεται από την πράξη με τον προτασιακό τύπο:
  - ◊ «Αν το bit  $i$  της δυαδικής τιμής του  $a$  είναι 0 και το bit  $i$  της δυαδικής τιμής του  $b$  είναι 0, τότε το bit  $i$  του αποτελέσματος είναι 0, διαφορετικά είναι 1. Η εφαρμογή της πρότασης για κάθε  $i$  δίνει τον τελικό επιστρεφόμενο αριθμό.»
- ❖ Ο  $\wedge$  ονομάζεται **ψηφιακό αποκλειστικό Η (bitwise exclusive OR)** και επιστρέφει μια ακέραια τιμή που αντιστοιχεί στο δυαδικό αριθμό που προέρχεται από την πράξη με τον προτασιακό τύπο:
  - ◊ «Αν το bit  $i$  της δυαδικής τιμής του  $a$  είναι ταυτόσημο με το bit  $i$  της δυαδικής τιμής του  $b$  είναι 0, τότε το bit  $i$  του αποτελέσματος είναι 0, διαφορετικά είναι 1. Η εφαρμογή της πρότασης για κάθε  $i$  δίνει τον τελικό επιστρεφόμενο αριθμό.»

# ΨΗΦΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (3/3)

- ❖ Οι ψηφιακοί τελεστές έχουν πολλές δυνατότητες. Ενδεικτικά, αν το bit  $i$  υπάρχει (1) ή όχι (0) στο στοιχείο  $i$  κάποιου συνόλου αναφοράς (πχ:  $\sim 0$ ), τότε
  - ◊ ο τελεστής  $\&$  αντιστοιχεί στην τομή των συνόλων
  - ◊ ο τελεστής  $|$  αντιστοιχεί στην ένωση των συνόλων
  - ◊ ο τελεστής  $\wedge$  αντιστοιχεί στη συμμετρική διαφορά των συνόλων (ένωση πλην την τομή)
- ❖ Οι όροι των τελεστών μπορούν να έχουν οκταδικές ή και δεκαεξαδικές τιμές, αν και πάντα οι ψηφιακοί τελεστές επιδρούν στις αντίστοιχες δυαδικές τιμές.

# Παράδειγμα

```
.....  
short i = 2, j = 5 , z;  
z = (i & j) | (i ^ j);  
printf (" z=%i ",z);  
.....
```

- ❖ Στο παράδειγμα μας, δηλώνονται οι short ακέραιοι i, j και z, όπου οι δύο πρώτοι έχουν αρχικές τιμές 2 και 5 αντίστοιχα, οι οποίες αντιστοιχούν στις δυαδικές τιμές 0010 και 0101.
- ❖ Η εντολή ισότητας αντικατάστασης θα επιστρέψει μια short ακέραια τιμή στη μεταβλητή z.
- ❖ Το δεξιό μέλος της εντολής θα εκτελεσθεί με βάση τον τελεστή |, από αριστερά στα δεξιά.
  - ◆ Άρα θα εκτελεσθεί πρώτα η πράξη (i&j), η οποία δίνει τη δυαδική τιμή 0000.
- ❖ Στην συνέχεια, θα εκτελεσθεί η πράξη (i^j), η οποία δίνει την δυαδική τιμή 0111.
- ❖ Τελικά η δυαδική τιμή 0000 θα έρθει σε επαφή με την τιμή 0111 δια μέσου του τελεστή |, με αποτέλεσμα να επιστραφεί η δυαδική τιμή 0111, η οποία αντιστοιχεί στον ακέραιο 7.
- ❖ Η τιμή 7 λοιπόν, θα εμφανιστεί στην οθόνη, σαν τιμή της μεταβλητής z.

# ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (1/2)

- ❖ Αναφέρονται μεταξύ δύο όρων με δράση από αριστερά προς δεξιά.
- ❖ Το επιστρεφόμενο αποτέλεσμα είναι τύπου int (0 ή 1).
- ❖ Συντακτικοί τύποι:

**a&&b , a||b**

- ❖ Ο **&&** ονομάζεται **λογικό ΚΑΙ (logical AND)**

- ❖ Αν η τιμή του a είναι μη μηδενική, εξετάζει την τιμή του b. Αν η τιμή του b είναι μη μηδενική τότε και μόνο τότε η επιστρεφόμενη τιμή είναι 1, διαφορετικά είναι μηδέν.
- ❖ Αν η τιμή του a είναι μηδενική, τότε η όλη πράξη δεν εξετάζει καν την τιμή του b και στην περίπτωση αυτή η επιστρεφόμενη τιμή είναι το μηδέν.

- ❖ Ο **||** ονομάζεται **λογικό Η (logical OR)**

- ❖ Αν η τιμή του a είναι μη μηδενική, τότε η πράξη δεν εξετάζει την καν την τιμή του b, επιστρέφοντας την τιμή 1.
- ❖ Αν η τιμή του a είναι μηδενική, τότε η πράξη εξετάζει την τιμή του b. Αν το b έχει μηδενική τιμή το επιστρεφόμενο αποτέλεσμα είναι μηδέν, διαφορετικά είναι 1.

# ΛΟΓΙΚΟΙ ΤΕΛΕΣΤΕΣ (2/2)

- ❖ Εξ ορισμού οι λογικοί τελεστές αδιαφορούν για το είδος των μεταβλητών  $a$  και  $b$ , εξετάζοντας μόνο αν έχουν μηδενικές ή όχι τιμές.
- ❖ Ωστόσο, σχεδόν όλοι οι μεταφραστές C++ απαγορεύουν να είναι τα άλλου τύπου από ακέραιοι, πραγματικοί ή δείκτες.
- ❖ Προσοχή, όταν εδώ αναφερόμαστε σε μηδενικές τιμές, εννοούμε το μηδέν όπως ορίζεται στο αντίστοιχο πεδίο ορισμού.
  - ◊ Παράδειγμα: το μηδέν των δεικτών ταυτίζεται με την έννοια του κενού δείκτη.

# Παράδειγμα

```
w=(x==y) && (z>=3.33); //1  
a=(p==0) || (&b[i]>&b[j]); //2  
c=((e&&f)|| (r!=g(k,&z))); //3
```

- ❖ Στο 1, αν η τιμή του  $x$  ταυτίζεται με την τιμή του  $y$ , τότε η επιστρεφόμενη τιμή στο  $w$  θα είναι 1, αν και μόνον αν, η τιμή του  $z$  είναι μεγαλύτερη ή και ίση από τη σταθερά 3.33.
  - ◊ Σε κάθε άλλη περίπτωση, το  $w$  θα πάρει την τιμή μηδέν.
- ❖ Στο 2, αν το  $p$  έχει μηδενική τιμή, τότε το  $a$  θα πάρει την τιμή 1 χωρίς καν να εξετασθεί η υπόλοιπη έκφραση.
  - ◊ Αν το  $p$  δεν έχει μηδενική τιμή, τότε θα εξετασθεί αν η διεύθυνση του στοιχείου  $b[i]$  είναι μεγαλύτερη από την διεύθυνση του στοιχείου  $b[j]$ .
  - ◊ Αν αυτό ισχύει, τότε θα επιστραφεί η τιμή 1 στο  $a$ , διαφορετικά θα επιστραφεί η τιμή μηδέν.
- ❖ Στο 3, θα εξετασθεί πρώτα η αριστερή παρένθεση.
  - ◊ Αν το  $e$  έχει τιμή μη μηδενική και το  $f$  επίσης, τότε θα επιστραφεί στο  $c$  η τιμή 1 χωρίς καν να εξετασθεί η δεύτερη παρένθεση.
  - ◊ Αν όμως το  $e$  ή το  $f$  ή και τα δύο, έχουν μηδενική τιμή, τότε θα εξετασθεί και η δεξιά παρένθεση.
  - ◊ Στην περίπτωση αυτή, αν η τιμή του  $r$  είναι διάφορη της τιμής που επιστρέφεται στην συνάρτηση  $g$ , τότε θα τεθεί το  $c$  ίσο με 1, διαφορετικά, θα επιστραφεί στο  $c$  η τιμή μηδέν.

# Παράδειγμα: έλεγχος δίσεκτου έτους

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

```
main()
{
    int year;
    cout << "Dwste to etos " << endl;
    cin >> year;
    if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))
        cout << "To etos " << year << " einai diseкто ";
    else
        cout << "To etos " << year << " den einai diseкто ";
}
```

Ένα έτος είναι δίσεκτο όταν

το έτος διαιρείται με το 4 αλλά όχι με το 100

το έτος διαιρείται με το 400

Χρήση λογικών τελεστών AND, OR, NOT

# Παράδειγμα: τριώνυμο

❖ Για την λύση της εξίσωσης 2ου βαθμού  $\alpha x^2 + \beta x + \gamma = 0$  χρησιμοποιούμε τη διακρίνουσα  $\Delta = \beta^2 - 4\alpha\gamma$

❖ Διακρίνουμε τις εξής περιπτώσεις:

◊ Αν  $\Delta > 0$ , το τριώνυμο έχει δύο πραγματικές ρίζες:

$$\rho_1 = \frac{-\beta + \sqrt{\Delta}}{2\alpha} \quad \rho_2 = \frac{-\beta - \sqrt{\Delta}}{2\alpha}$$

◊ Αν  $\Delta = 0$ , το τριώνυμο έχει μία «διπλή» πραγματική ρίζα:

$$\rho = \frac{-\beta}{2\alpha}$$

◊ Αν  $\Delta < 0$ , το τριώνυμο δεν έχει πραγματικές ρίζες.



# Παράδειγμα: τριώνυμο

```
<include <iostream>
<include <math.h>
using namespace std;
main()
{
    float a, b, c, d;
    cin >> a >> b >> c >>;
    if (a==0 && b==0 && c==0) cout << "H exisosi einai aoristi" <<endl;
    else if (a==0 && b==0 && c!=0) cout << "H exisosi einai adynati" <<endl;
    else if (a==0 && b!=0) cout << "H exisosi exei ti mondaiki lisi " << -c/b <<endl;
    else
    {
        d=b*b-4*a*c;
        if (d<0) cout << "H exisosi exei dio rizes migadikes " << endl;
        else if (d==0) cout <<"H exisosi exei mondaiki lisi tin " << -b/(2*a));
        else cout <<"H exisosi exei dyo rizes " << (-b+sqrt(d))/(2*a), (-b-sqrt(d))/(2*a));
    }
}
```

# ΣΕΙΡΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (1/2)

- ❖ Υπάρχουν συντακτικοί τύποι που επιτρέπουν να είναι παρούσα μια μόνο έκφραση, όπως πχ. μια παράμετρος συνάρτησης μέσα στη λίστα παραμέτρων της.
- ❖ Σε άλλες όμως περιπτώσεις (όχι πολύ συνηθισμένες) βολεύει προγραμματιστικά να υπάρχουν περισσότερες εκφράσεις.

Σε αυτές χρησιμοποιείται ο **σειριακός τελεστής** που συμβολίζεται με κόμμα (,) και έχει γενικό τύπο:

**(p1,p2)**

- ◊ όπου εκτελείται πρώτα η έκφραση p1, στη συνέχεια η p2 και το αποτέλεσμα που επιστρέφεται έχει τον τύπο του p2 καθώς και την τιμή του.

# ΣΕΙΡΙΑΚΟΙ ΤΕΛΕΣΤΕΣ (2/2)

- ❖ Προσοχή, δεν πρέπει να συγχέεται το απλό κόμμα που χρησιμοποιείται σε δηλώσεις ή και σε λίστες παραμέτρων, με το κόμμα του τελεστή.

❖ Παράδειγμα: `sum ( (x=3 , y-x) , w , z )`

καλείται η συνάρτηση `sum` όπου περνά κανονικά η τιμή της μεταβλητής `z`, η τιμή της `w` και η τιμή `y-x` αφού εκτελεστεί πρώτα η εντολή αντικατάστασης `x=3`.

- ❖ Μια άλλη, επίσης σπάνια, εφαρμογή εμφανίζεται μέσα στα πλαίσια της εντολής `for`, η οποία μέσα στις εκφράσεις της δέχεται το κόμμα. Εάν το επιθυμούμε, μπορούμε να θέσουμε μια από αυτές σαν μια αλυσίδα εκφράσεων με τη βοήθεια του σειριακού τελεστή.

❖ Παράδειγμα: αν δοθεί

```
for (i=1; i+j<20; i++, j++)  
    printf("%i ", i);
```

σε κάθε επανάληψη της διαδικασίας, η τιμή του `i` θα αυξάνεται κατά 1 και στη συνέχεια θα συμβαίνει το ίδιο και με την τιμή του `j`.

# ΥΠΟ ΣΥΝΘΗΚΗ ΤΕΛΕΣΤΗΣ (1/2)

- ❖ Ο υπό συνθήκη τελεστής (**conditional operator**) είναι ένας τριαδικός τελεστής με συντακτικό τύπο:

**Υπόθεση α ? έκφραση β : έκφραση γ**

- ◊ Αν η τιμή της υπόθεσης α είναι μη μηδενική (true), εκτελείται η έκφραση β, διαφορετικά εκτελείται η γ.  
Επομένως εκτελείται μόνο μία έκφραση, η τιμή της οποίας επιστρέφεται σαν τιμή της όλης παράστασης (πράξης).

- ❖ Παράδειγμα 1:     **`minab=a<b?a:b;`**

- ◊ Αν  $a < b$  η επιστρεφόμενη τιμή είναι α, διαφορετικά το b.
- ◊ Η επιστρεφόμενη τιμή σχετίζεται με τη μεταβλητή

- ❖ Παράδειγμα 2:     **`m=i<0?-i:i;`**

- ◊ όπου η μεταβλητή m θα είναι τελικά η απόλυτη τιμή της τιμής του i.

# ΥΠΟ ΣΥΝΘΗΚΗ ΤΕΛΕΣΤΗΣ (2/2)

- ❖ Άδικα ορισμένοι υποστηρίζουν ότι ο υπό συνθήκη τελεστής ισοδυναμεί με την εντολή if-else:

```
if υπόθεση α
    έκφραση β ;
else
    έκφραση γ ;
```

Ο τελεστής είναι ταχύτερος και η ταχύτητα δράσης του είναι ασύγκριτη σε σχέση με αντίστοιχη εντολή!

- ❖ Ο υπό συνθήκη τελεστής δέχεται και φαινόμενο nesting (να χρησιμοποιηθεί και με άλλους τέτοιους τελεστές).
- ❖ Η υπόθεση α πρέπει να είναι ακέραιου τύπου, δεκαδικού ή δείκτης.
- ❖ Οι εκφράσεις β και γ μπορεί ακόμη να είναι ενώσεις, δομές ή δείκτες.

# ΠΑΡΑΔΕΙΓΜΑ

```
#include <math.h>
#include <iostream>

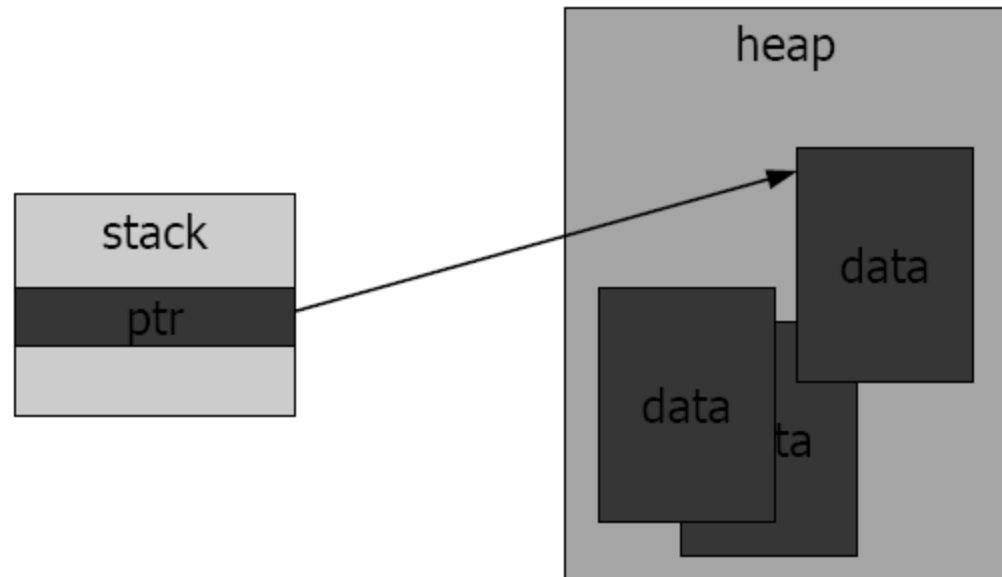
using namespace std;

int main()
{
    float a,b,c,d;
    cout <<"Dwse tous sintelestes triwnimou a, b, c"<<endl;
    cin >> a >> b >> c;
    d=b*b-4*a*c;
    (d>=0)?cout<<(-b+sqrt(d))/(2*a) <<(-b-sqrt(d))/(2*a): cout<<"mi pragmatikes
    rizes..."<<endl;

    return 0;
}
```

# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (1/6)

- ❖ Υπάρχουν δύο τελεστές, οι \* και &, και αφορούν προσπέλαση τιμής έμμεσα και αντίστοιχης διεύθυνσης με τη βοήθεια pointers.
- ❖ Γενικός συντακτικός τύπος του τελεστή \* : \*α
  - ◊ όπου α είναι δείκτης, και
  - ◊ το επιστρεφόμενο αποτέλεσμα είναι η τιμή που δείχνει ο δείκτης.  
Προσοχή: Αν ο δείκτης είναι ο κενός δείκτης (null pointer), τότε το αποτέλεσμα είναι απροσδιόριστο.



# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (2/6)

- ❖ Γενικός συντακτικός τύπος του τελεστή & : &a
  - ◇ όπου a είναι μεταβλητή, και
  - ◇ το επιστρεφόμενο αποτέλεσμα είναι η διεύθυνση της τιμής της μεταβλητής a μέσα στη μνήμη.  
Προσοχή: Ο & δεν έχει νόημα σε μέλη δομής με bits ή σε μεταβλητές κατηγορίας register.
- ❖ Σε κάθε σχεσιακό τελεστή οι δύο όροι επιτρέπεται να είναι δείκτες του ίδιου τύπου.
  - ◇ Οι τελεστές δεν έχουν καμία πρακτική σημασία αν και οι δύο δείκτες δεν αφορούν το ίδιο πεδίο μνήμης.
  - ◇ Παράδειγμα: έχει νόημα η σχεσιακή σύγκριση μεταξύ δεικτών που αφορούν στοιχεία της ίδιας μήτρας διότι αναφέρονται στο ίδιο πεδίο μνήμης.
- ❖ Με τους τελεστές == και != μπορούμε να ελέγχουμε αν ένας δείκτης είναι κενός (null pointer) το οποίο μπορεί να γίνει με αρχική τιμή ή και με ισότητα αντικατάστασης.
  - ◇ Σε αυτή την περίπτωση επιτρέπεται ο πρώτος όρος να είναι δείκτης και ο δεύτερος η σταθερά μηδέν (0).



# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (3/6)

## ❖ Παράδειγμα:

```
int *p, i, m[10];  
p=&m[1];  
i=*p;
```

◊ όπου ο δείκτης *p* περιέχει τη διεύθυνση της τιμής του στοιχείου *m[1]* και η ακέραια μεταβλητή *i* την τιμή που δείχνεται από το δείκτη *p*, δηλαδή την τιμή του *m[1]*.

❖ Μεταξύ των τελεστών κατεύθυνσης και διεύθυνσης η τιμή ενός *x* ταυτίζεται με το αποτέλεσμα που επιστρέφεται από την έκφραση *(&x)*, δηλαδή από την τιμή που υπάρχει στη διεύθυνση για τη μεταβλητή *x*.

❖ Υπενθυμίζεται ότι το όνομα μιας μήτρας, αν χρησιμοποιηθεί σαν δείκτης, ισοδυναμεί με τη διεύθυνση του πρώτου στοιχείου της.

# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (4/6)

## ❖ Παράδειγμα:

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

```
int main ()                                firstvalue is 10, secondvalue is 20
{
    int firstvalue, secondvalue;
    int *mypointer;
    mypointer = &firstvalue;
    *mypointer = 10;
    mypointer = &secondvalue;
    *mypointer = 20;
    cout << "firstvalue is " << firstvalue << '\n';
    cout << "secondvalue is " << secondvalue << '\n';
    return 0;
}
```

# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (5/6)

## ❖ Παράδειγμα:

```
int ival = 1024;
int *pi = 0; // pi initialised to address no object
int *pi2 = &ival; // pi2 initialised to address of ival
int *pi3; //ok but dangerous, pi3 is uninitialised
pi = pi2;
pi2 = 0;
```

# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ΔΙΕΥΘΥΝΣΗΣ (6/6)

```
#include <iostream>
using namespace std;
main()
{
    int i, j;
    i=5;
    j=i;
    cout << j << '\n' << i << '\n'; // j=5 i=5
    cout << &j << '\n' << &i << '\n'; // different addresses

    int ii=3;
    int &jj = ii;
    cout << jj << '\n' << ii << '\n'; // jj=3, ii=3
    cout << &jj << '\n' << &ii << '\n'; // same addresses

    int *p, *q;
    p = &ii;
    q = &jj;

    cout << p << '\n' << q << '\n'; // same addresses
}
```

# ΤΕΛΕΣΤΕΣ ΚΑΤΕΥΘΥΝΣΗΣ & ARRAYS

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

int main ()
{
    int numbers[5];
    int * p;
    p = numbers; *p = 10;
    p++; *p = 20;
    p = &numbers[2]; *p = 30;
    p = numbers + 3; *p = 40;
    p = numbers; *(p+4) = 50;
    for (int n=0; n<5; n++)
        cout << numbers[n] << ", "; //10, 20, 30, 40, 50
    return 0;
}
```

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (1/12)

- ❖ Όταν η τιμή μιας έκφρασης ή μεταβλητής ή σταθεράς σημειώνεται σαν τιμή σε μια μεταβλητή, τότε έχουμε **πράξη αντικατάστασης (assignment)**.
  - ◊ Προσοχή, στις γλώσσες προγραμματισμού η έννοια της αντικατάστασης εκφράζεται κατά κανόνα με εντολή (statement) και όχι με τελεστή, εκτός του λογικού προγραμματισμού (Prolog) όπου η έννοια αυτή είναι πολύπλοκη.
  - ◊ Η C++ αποτελεί εξαίρεση και χρησιμοποιεί ειδικούς τελεστές για την εργασία αυτή, γεγονός που αριστοποιεί τις ταχύτητες αντικατάστασης και που προσφέρει περισσότερες δυνατότητες στην κωδικοποίηση πολύπλοκων αλγορίθμων.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (2/12)

❖ Γενικός τύπος πράξης αντικατάστασης:

**όνομα\_μεταβλητής Τελεστής\_αντικατάστασης έκφραση**

- ❖ Η τιμή της έκφρασης, αφού επεξεργασθεί από τον τελεστή, τίθεται σαν μεταβλητή στην τιμή του αριστερού μέλους.
- ❖ Αν η τιμή είναι διαφορετική από το είδος τιμών της μεταβλητής, τότε παράλληλα γίνεται μετατροπή της τιμής στο είδος τιμών της μεταβλητής.
- ❖ Η πράξη της αντικατάστασης εκτελείται από τα δεξιά προς τα αριστερά ακόμη και σε πολύπλοκες καταστάσεις.

❖ Οι τελεστές αντικατάστασης χωρίζονται σε δύο κατηγορίες:

- ❖ **Απλή αντικατάσταση**
- ❖ **Σύνθετη αντικατάσταση**

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (3/12)

## Απλή αντικατάσταση

### A. Τελεστής αύξησης ή μείωσης τιμής κατά 1

#### ❖ ΣΥΝΤΑΚΤΙΚΟΙ ΤΥΠΟΙ:

<b>++x</b>	<b>x++</b>
<b>- x</b>	<b>x-</b>

- ❖ Αν το ++ ή το - - είναι αριστερά της μεταβλητής ή έκφρασης x, τότε πρώτα αυξάνεται ή μειώνεται το x κατά 1 και στη συνέχεια χρησιμοποιείται η νέα προκύπτουσα τιμή του x από άλλους τελεστές της παράστασης.
- ❖ Αν το ++ ή το - - είναι δεξιά του x, πρώτα χρησιμοποιείται η παρούσα τιμή του x από άλλους τελεστές της παράστασης και στη συνέχεια αυξάνεται ή μειώνεται η τιμή του x κατά 1.
- ❖ Το x μπορεί να είναι integer, float, ή τύπου pointer. Στην τελευταία περίπτωση η αύξηση κατά 1 του x δηλώνει (δείχνει) το επόμενο αντικείμενο, ενώ η μείωση κατά 1 το προηγούμενο, σύμφωνα πάντα με τη σειρά διάταξης.



# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (4/12)

## Απλή αντικατάσταση

❖ Παράδειγμα: Έστω οι ακόλουθες παραστάσεις

$$(++a) * (--b) \quad (a++) * (b--) \quad (++a) * (b++)$$

Εάν  $a=b=2$ , επιστρέφουν τις τιμές 3,4 και 6 αντίστοιχα.

- ❖ Στην πρώτη παράσταση πρώτα αυξάνεται το  $a$  κατά 1, μειώνεται το  $b$  κατά 1 και στη συνέχεια εκτελείται ο πολλαπλασιασμός.
- ❖ Στη δεύτερη παράσταση, πρώτα εκτελείται ο πολλαπλασιασμός μεταξύ των  $a$ ,  $b$  και στη συνέχεια αυξάνεται το  $a$  κατά 1 και μειώνεται το  $b$  κατά 1.
- ❖ Στην τρίτη παράσταση πρώτα αυξάνεται το  $a$  κατά 1, γίνεται ο πολλαπλασιασμός μεταξύ της νέας τιμής του  $a$  και της τιμής του  $b$  και στη συνέχεια αυξάνεται το  $b$  κατά 1.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (5.1/12)

## Παράδειγμα

Κατά την αύξηση/μείωση των δεικτών που αντιστοιχούν σε array, το αποτέλεσμα αναφέρεται στα στοιχεία του array (κι όχι, π.χ. στις διευθύνσεις). Για παράδειγμα,

```
int p[] = {1,2,3,4};
```

```
int* q = p; cout << *q << endl;
```

```
q++; cout << *q << endl;
```

μετά την αύξηση, ο q δείχνει στον επόμενο ακέραιο (του array) κι όχι στην επόμενη θέση μνήμης

◇  $*++q = *(++q)$

◇  $++*q = ++(*q)$

◇  $*q++ \neq (*q)++$

◇  $*(q++) = *q++$

# Παράδειγμα

```
#include <iostream.h>

main()
{
    int p[] = {10,20,30,40};
    int *q = p;
    int x;

    x = *q; // x equals to the value of the integer that q points to (x=10)
    cout << x << endl; // number 10 is printed

    x = *q++; // x equals to the value of the integer that q points to (x=10), then q points to the next
    integer, i.e. p[1]
    cout << x << endl; // number 10 is printed

    x = *(q++); // x equals to the value of the integer that q points to (x=20), then q points to the
    next integer, i.e. p[2]
    cout << x << endl; //number 20 is printed

    x = (*q)++; // x equals to the value of the integer that q points to (x=30), then the value of p[2]
    is incremented by 1. No change for q
    cout << x << endl; // number 30 is printed

    x = *q; // x equals to the value of the integer that q points to (x=31)
    cout << x << endl; // number 31 is printed

    x = *++q; // first increment q to point to the next integer, i.e. p[3], and then assign to x its
    value (x=40)
    cout << x << endl; // number 40 is printed

    x = ++*q; // first increment the value of the integer that q points to and then assign this value to
    x (x=41)
    cout << x << endl; // number 41 is printed
}
```

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (5/12)

## Απλή αντικατάσταση

### B. Τελεστής απλής αντικατάστασης τιμής

❖ ΣΥΝΤΑΚΤΙΚΟΣ ΤΥΠΟΣ:

=

- ❖ Σημειώνει την τιμή της δεξιάς έκφρασης σαν τιμή στη μεταβλητή του αριστερού μέλους.
- ❖ Η τιμή είναι πάντοτε του είδους τιμής των μεταβλητών.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (6/12)

## Απλή αντικατάσταση

❖ Παράδειγμα: Αν δοθούν οι εντολές

```
double x,y;  
int z;  
z=1;  
x=y=z;
```

◊ Το z παίρνει την τιμή 1, ενώ τα x, y παίρνουν την τιμή 1.0 με απαιτήσεις διπλής ακρίβειας, δηλ. σε απεικόνιση 8 bytes κύριας μνήμης (για τους περισσότερους υπολογιστές).

Προσοχή στην τελευταία εντολή όπου σημειώνεται πολλαπλή ισότητα αντικατάστασης, που εκτελείται από δεξιά προς αριστερά.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (7/12)

## Σύνθετη αντικατάσταση

❖ Ο **τελεστής σύνθετης αντικατάστασης** απλά εκτελεί την απαιτούμενη πράξη που του αντιστοιχεί και στη συνέχεια εκτελείται η πράξη της απλής αντικατάστασης τιμής.

❖ Γενικός συντακτικός τύπος:

**μεταβλητή τελεστής = έκφραση**

που ισοδυναμεί πλήρως με τον ακόλουθο συντακτικό τύπο:

**μεταβλητή = μεταβλητή τελεστής έκφραση**

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (8/12)

## Σύνθετη αντικατάσταση

❖ Παράδειγμα: Αν δοθεί η εντολή

$x+=y;$

ισοδυναμεί με την εντολή

$x=x+y;$

Προσοχή, οι δύο εντολές δεν είναι τελείως ισοδύναμες από πλευράς υπολογιστικού χρόνου, καθώς στη δεύτερη εκτελούνται διπλάσια υπολογιστικά βήματα από την πρώτη.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (9/12)

- ❖ \*= Πολλαπλασιαστική αντικατάσταση
- ❖ /= Διαιρετική αντικατάσταση
- ❖ %= Υπολοίπου αντικατάσταση
- ❖ += Προσθετική αντικατάσταση
- ❖ -= Αφαιρετική αντικατάσταση
- ❖ >>= Δεξιάς μετακίνηση αντικατάσταση
- ❖ <<= Αριστερής μετακίνηση αντικατάσταση
- ❖ &= Ψηφιακή AND αντικατάσταση
- ❖ |= Ψηφιακή OR αντικατάσταση
- ❖ ^= Ψηφιακή OR αποκλειστική αντικατάσταση



# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (10/12)

## Σύνθετη αντικατάσταση

❖ Έστω η συνάρτηση fun:

```
fun(i+2, i=j+1);
```

❖ Στην περίπτωση αυτή, η C++ δεν προσδιορίζει καμία σειρά προσδιορισμού των τιμών της λίστας παραμέτρων.

Είναι εξίσου πιθανό να υπολογισθεί πρώτα το  $i+2$ , αλλά και να υπολογισθεί πρώτα το  $i=j+1$ !

❖ Αν δοθεί η εντολή

```
x=y++=z++;
```

❖ Αν η μηχανή στο στάδιο αυτό γνωρίζει  $z=0$  τότε η τιμή του  $x$  μπορεί να είναι 0, μπορεί όμως και 1.

❖ Κανονικά πρέπει η τιμή  $z=0$  να σημειωθεί στο  $y$  και η τιμή αυτή στο  $x$ , οπότε  $x=0$ , και στη συνέχεια να αυξηθεί η τιμή των  $y$  και  $z$  κατά 1.

❖ Ωστόσο, αρκετοί επεξεργαστές θα εκτελούσαν πρώτα την αντικατάσταση  $y++=z++$ , οπότε το  $y$  θα είχε την τιμή 1, και στη συνέχεια την αντικατάσταση  $x=y++$ , με επακόλουθο το  $x$  να ισούται με 1 σε τελική φάση.

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (11/12)

❖ Παράδειγμα με τελεστές αντικατάστασης:

```
.....  
int x,y,z,w;  
y=z=1;  
x=z++;  
w=++y;  
w*=x;  
w+=w;  
printf("x=%d y=%d z=%d w=%d", x, y, z, w);  
.....
```

◊ Μια τυπική έξοδος είναι η:

```
x=1 y=2 z=2 w=4
```

# ΤΕΛΕΣΤΕΣ ΑΝΤΙΚΑΤΑΣΤΑΣΗΣ (12/12)

- ◇ Στην αρχή εφαρμόζεται η πολλαπλή ισότητα:

**y=z=1 ;**

όπου η τιμή των z και y γίνονται 1

- ◇ Έπειτα εφαρμόζεται η απλή αντικατάσταση:

**x=z++ ;**

όπου το x παίρνει την τιμή του y, δηλ. 1, και στη συνέχεια η τιμή του z αυξάνεται κατά 1 και γίνεται 2.

- ◇ Αμέσως μετά εκτελείται η απλή αντικατάσταση:

**w=++y ;**

όπου αυξάνεται η τιμή του y κατά 1, και γίνεται 2, και στη συνέχεια η τιμή αυτή δίνεται στο w.

- ◇ Τέλος εκτελούνται οι σύνθετες αντικαταστάσεις:

**w\*=x ;** (δηλαδή **w=w\*x ;**)

**w+=w ;** (δηλαδή **w=w+w ;**)

οπότε η πρώτη θέτει σαν τιμή του w την  $2*1=2$ , ενώ η δεύτερη θέτει τελικά σαν τιμή στο w την τιμή  $2+2=4$ .

# ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (1/2)

- ❖ Όταν διαφορετικοί τύποι δεδομένων συνυπάρχουν μέσα σε εκφράσεις ή και σε πράξεις αντικατάστασης, τότε γίνεται αυτόματα η συνηθισμένη μετατροπή, αρκεί να έχει νόημα μια τέτοια συνύπαρξη.
- ❖ Παράδειγμα: το άθροισμα ενός float με ένα int θα είναι float, ενώ αν ο δείκτης μιας μήτρας είναι double τότε δεν είναι δυνατό να γίνει αυτόματη μετατροπή.
- ❖ Γενικά στη C++, ο Αναλυτής έχει την κύρια ευθύνη μετατροπών, και όχι τόσο ο μεταγλωττιστής.

# ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (2/2)

❖ Υπάρχουν δύο βασικοί τρόποι μετατροπής τιμών:

◊ (implicit cast) Αυτόματα από το μεταγλωττιστή της C++

◊ (explicit cast) Με ρητή μετατροπή τύπου, όπου μια έκφραση `expr` μετατρέπεται σε τιμή τύπου `typename` με τον απλό τύπο:

**`(typename) expr`**

Στην ουσία η νέα προκύπτουσα τιμή της έκφρασης `expr` σημειώνεται σε μια μεταβλητή την οποία μπορεί να χρησιμοποιήσει ο Αναλυτής στη συνέχεια με απλή αντικατάσταση:

**`var = (typename) expr`**

Παράδειγμα: Αν `fvar` είναι `double` και `ivar` είναι `int`, τότε αν δοθεί

**`fvar = (double) ivar`**

η μεταβλητή `fvar` θα έχει την τιμή της `ivar` αλλά μετασχηματισμένη σε τιμή διπλής ακρίβειας (πχ. από 2 bytes έκταση σε 8 bytes).

# ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (1/4)

❖ Στις αυτόματες μετατροπές η C++ εφαρμόζει **απλούς κανόνες**:

[K1] Τύποι char και short μετατρέπονται σε int

[K2] Ο τύπος float μετατρέπεται σε double

[K3] Αν μετά την εφαρμογή των κανόνων [K1], [K2] ο ένας από τους δύο όρους είναι double, το αποτέλεσμα είναι double

[K4] Αν μετά την εφαρμογή των κανόνων [K1], [K2] ο ένας από τους δύο όρους είναι long, το αποτέλεσμα είναι long, εκτός αν έχει ήδη βρει εφαρμογή ο κανόνας [K3]

[K5] Αν μετά την εφαρμογή των κανόνων [K1], [K2] δεν εφαρμοστεί ο κανόνας [K3] και [K4], τότε αν ο όρος είναι τύπου unsigned, το αποτέλεσμα είναι επίσης τέτοιου τύπου

[K6] Αν μετά την εφαρμογή των κανόνων [K1], [K2] δεν εφαρμοστεί κάποιος από τους κανόνες [K3], [K4], [K5] τότε το αποτέλεσμα είναι πάντα τύπου int

# ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (2/4)

- ❖ Οι κανόνες αυτόματης μετατροπής τιμών [K1]-[K6] αποτελούν στην ουσία ένα μικρό **αλγόριθμο** ο οποίος μπορεί να γραφεί ως εξής:

**Εφαρμογή** του [K1]

**Εφαρμογή** του [K2]

**Αν ισχύει** ο [K3], **τότε** το αποτέλεσμα είναι double

**Διαφορετικά**

**Αν ισχύει** ο [K4], **τότε** το αποτέλεσμα είναι long

**Διαφορετικά**

**Αν ισχύει** ο [K5], **τότε** το αποτέλεσμα είναι unsigned

**Διαφορετικά**

Το αποτέλεσμα είναι int

# ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (3/4)

- ❖ Παράδειγμα: Αν  $a$  είναι `char`,  $b$  `short` και  $f$  `float` και δοθεί η έκφραση  
 $(a+b) * f$ 
  - ◇ Αρχικά θα εφαρμοστεί ο κανόνας [K1] και η τιμή  $(a+b)$  θα είναι `int`
  - ◇ Στη συνέχεια θα εφαρμοστεί ο [K2] και το  $f$  θα είναι στην ουσία `double`
  - ◇ Τέλος θα εφαρμοστεί ο [K3] με αποτέλεσμα η τελική τιμή της έκφρασης να είναι `double`



# ΕΠΙΚΙΝΔΥΝΕΣ ΜΕΤΑΤΡΟΠΕΣ ΤΙΜΩΝ (4/4)

Είδος τιμής A	Είδος τιμής B	Παρατηρήσεις
float	long	Η τιμή γίνεται float. Πιθανή αλλαγή τιμής.
double	long	Η τιμή γίνεται double. Πιθανή αλλαγή τιμής.
long	float	Η τιμή γίνεται long. Πιθανόν, απροσδιόριστο αποτέλεσμα.
long	double	Η τιμή γίνεται long. Πιθανόν, απροσδιόριστο αποτέλεσμα.
float	double	Η τιμή γίνεται float. Πιθανή αλλαγή τιμής στην ακρίβεια ή και απροσδιόριστο αποτέλεσμα.
οτιδήποτε	void	Απαγορεύεται.
structure	union	Απαγορεύεται
union	structure	Απαγορεύεται
enum	int	Επιτρέπεται. Προσοχή αν η ακέραια τιμή είναι long.
int	enum	Επιτρέπεται. Προσοχή αν η ακέραια τιμή είναι short.
pointer	int	Επιτρέπεται, αν ο pointer είναι int.
int	pointer	Επιτρέπεται, αν ο pointer είναι int.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (1/6)

❖ Υπάρχουν τρεις κατηγορίες διαχείρισης μνήμης στα προγράμματα C++

1. Ο **Linker** κανονίζει τη **στατική μνήμη (static memory)**.

- ❖ Διευθετείται μέσα στη μνήμη ο κώδικας, καθολικές μεταβλητές, στατικά μέλη τάξεων, στατικές μεταβλητές συναρτήσεων, κλπ.
- ❖ Όταν το διαμέρισμα μνήμης αντικειμένων είναι σε στατική μνήμη, γίνεται μια φορά και διαρκεί μέχρι τον τερματισμό του προγράμματος.

2. Η by default αντίδραση της C++ είναι η **αυτόματη μνήμη (automatic memory)**,

- ❖ Όλες οι τοπικές μεταβλητές, υποκατάστατες παράμετροι συναρτήσεων και γενικά κάθε τι που στη C++ χαρακτηρίζεται σαν αυτόματο (auto) αντικείμενο, απαιτούν αυτόματα μνήμη και στη συνέχεια, όταν πάψει η ορατότητά τους, την ελευθερώνουν.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (2/6)

## 3. Η ελεύθερη μνήμη (**free store**) ή απλά **heap memory** (δυναμική μνήμη ή σωρός).

- ◊ Ο εκτελέσιμος κώδικας ζητά μνήμη στη διάρκεια της εκτέλεσης και αργότερα την ελευθερώνει.
- ◊ Η δέσμευση μνήμης γίνεται για αντικείμενα με τη βοήθεια ειδικών τελετών ή συναρτήσεων. Με αντίστοιχο τρόπο γίνεται και η απελευθέρωση.
- ◊ Στη C++ και στη Visual C++ καλό είναι η κατανομή τέτοιας μνήμης να γίνεται με τον τελεστή `new` και η απελευθέρωση με τον τελεστή `delete`.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (3/6)

- ❖ Ο τελεστής `new`
  - ◇ με απλό αντικείμενο δημιουργεί δυναμικά ένα δείκτη προς το αντικείμενο αυτό.
  - ◇ σε μήτρες δημιουργεί δυναμικά ένα δείκτη προς το πρώτο αντικείμενο της μήτρας.
  - ◇ επιστρέφει μηδενική τιμή όταν η πράξη αποτύχει
- ❖ Η μνήμη που δεσμεύεται με τον `new` μπορεί να απελευθερωθεί μόνο με τον τελεστή `delete` και μόνο μέσα στην ορατότητα του αντικειμένου που ορίσθηκε.
- ❖ διαφορετικά, το `delete` δεν μπορεί να δράσει και η δεσμευμένη τιμή από τον `new` μένει μέχρι το τέλος της εκτέλεσης του προγράμματος.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (4/6)

❖ ΣΥΝΤΑΚΤΙΚΟΪ ΤΥΠΟΥ ΤΕΛΕΣΤΩΝ new και delete:

**new(λίστα τιμών διευθύνσεων)τύπος(λίστα αρχικών τιμών)**

**delete(δείκτης) ή delete[ ] δείκτης**

❖ Η **λίστα τιμών διευθύνσεων** και η **λίστα αρχικών τιμών** μπορούν να απουσιάζουν από το συντακτικό τύπο του τελεστή new.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (5/6)

## ❖ Παράδειγμα:

```
double *x=new double;  
float *y=new float (33.123);  
int *z=new (0x0040) int;  
char *array1=new char [10000];  
float (*point) [50] [80];  
point=new float [40] [50] [80];
```

### ◊ Οπού ορίζεται δυναμικά

- ◆ η μεταβλητή x τύπου pointer to double
- ◆ η y τύπου pointer to float με αρχική τιμή 33.123
- ◆ η z τύπου pointer to int με διαμέρισμα μνήμης στη διεύθυνση 0x0040
- ◆ η μήτρα χαρακτήρων array1 με 10.000 στοιχεία και,
- ◆ η πολυδιάστατη μήτρα point με τιμές float και διαστάσεις 40×50×80.

# ΤΕΛΕΣΤΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΜΝΗΜΗΣ (6/6)

- ◊ Όλα τα αντικείμενα μπορούν στη συνέχεια να διαγραφούν με παράλληλη ελευθέρωση της μνήμης τους, με τις ακόλουθες εντολές:

```
delete x;  
delete y;  
delete z;  
delete []array1;  
delete []point;
```

# ΠΑΡΑΔΕΙΓΜΑ

```
void main()
{
    char *first = new char[5000];
    for (int i=0; i<5000; i++)
    {
        if (i==1000)
        {
            int (*second) [500];
            second = new int[100] [500];
        }
        .....
    }
    delete [] first;
    delete [] second;
    ...
}
```

Η μήτρα second  
δεν θα διαγραφεί!



# ΠΑΡΑΔΕΙΓΜΑ

```
#include <iostream>
using namespace std;
int g = 1000;
void p(int i)
{
    int* j = new int(30); // now j points to heap
    cout << "i in p is : " << i << endl;
    cout << "j in p points to : " << *j << "\n" <<
    endl;
    delete j;
}
void f(int i)
{
    int j = 20;
    static int k = 100;
    cout << "i in f is : " << i << endl;
    cout << "j in f is : " << j << endl;
    cout << "k in f is : " << k << endl;
    cout << "g in f is : " << g << "\n" << endl;
    i++; j++; k++; g++;
    p(i);
}
```

```
int main()
{
    int i = 5;
    cout << "i in main is :" << i << '\n' << endl;
    f(i);
    cout << "i in main is :" << i << '\n' << endl;
    f(i++);
    cout << "i in main is :" << i << '\n' << endl;
    f(++i);
    cout << "i in main is :" << i << '\n' << endl;
    cout << "g in main is :" << g << '\n' << endl;
    return 0;
}
```

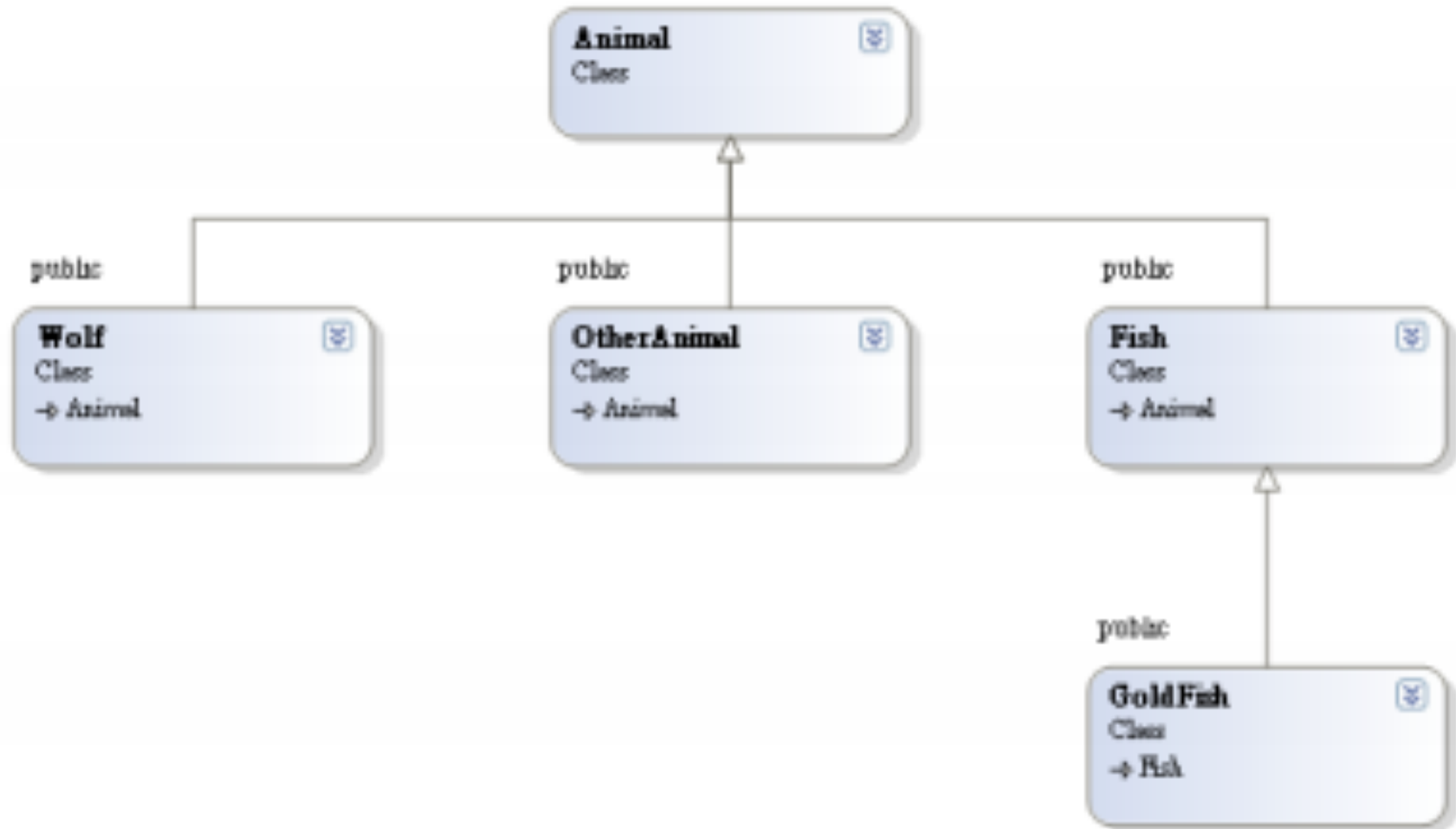
# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (1/5)

- ❖ Όταν μια κλάση βασίζεται σε μια άλλη **βασική κλάση**, τότε ονομάζεται **παράγωγη κλάση**. Σε αυτή της περίπτωση η δήλωση της δεύτερης βασίζεται στο public μέρος της πρώτης:  
**class παράγωγη: public βασική**
- ❖ Όταν μια βασική κλάση χρησιμοποιεί συναρτήσεις που επιτρέπεται στον αναλυτή να ορίσει ξανά μέσα στις παράγωγες κλάσεις, τότε αυτές οι συναρτήσεις καλούνται **εικονικές (virtual functions)** και ο compiler εγγυάται για την ορθή αντιστοίχιση μεταξύ αντικειμένων και συναρτήσεων.
  - ◊ Η δήλωση μιας εικονικής συνάρτησης γίνεται με τη λέξη-κλειδί **virtual**.
  - ◊ Δεν πρέπει να υπάρχουν σημαντικές διαφορές στις παραμέτρους εισόδου/εξόδου μιας τέτοιας συνάρτησης από την ανάλυσή της από μια παράγωγη κλάση σε άλλη.

# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (2/5)

- ❖ Στον αντικειμενοστραφή προγραμματισμό, η δυνατότητα ορθής συμπεριφοράς των συναρτήσεων μέσα στις κλάσεις, ονομάζεται **Πολυμορφισμός (Polymorphism)**.
  - ◊ Ένας πολυμορφικός τύπος χρησιμοποιεί εικονικές συναρτήσεις.
  - ◊ Ο χειρισμός των αντικειμένων γίνεται με δείκτες και διευθύνσεις (αναφορές).

# Παράδειγμα



# Παράδειγμα

```
class Animal{  
    void /*nonvirtual*/ move() {  
        cout << "This animal moves in some way" << endl;  
    }  
    virtual void eat() {}  
};
```

// The class "Animal" may possess a declaration for eat() if desired.

```
class Llama : public Animal{  
    // The non virtual function move() is inherited but cannot be overridden  
    void eat() {  
        cout << "Llamas eat grass!" << endl;  
    }  
};
```

# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (2/5)

- ❖ Παράδειγμα: Βασική κλάση με εικονική συνάρτηση

```
//Βασική κλάση
class Base_class
{
    public
        virtual void fun1();    //εικονική
        void fun2();           //μη εικονική
};
```

- ◈ Η ανάπτυξη της συνάρτησης fun2 έχει νόημα μέσα στα πλαίσια της κλάσης Base\_class ενώ η **ανάπτυξη της fun1 έχει νόημα μόνο μέσα σε άλλες παραγόμενες κλάσεις.**

# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (3/5)

- ❖ Θεωρούμε έναν **πολυμορφισμό** με τον προσδιορισμό της παραγόμενης κλάσης `Kid_class` από τη βασική συνάρτηση `Base_class`:

```
//Προσδιορισμός παραγόμενης κλάσης
class Kid_class : public Base_class
{
public:
    void fun1();           //Είναι η εικονική της βάσης
    void fun2();           //Απλή τοπική συνάρτηση
};
//Τα σώματα των fun1, fun2 μέσα στην παραγόμενη κλάση
void Kid_class::fun1()
    {cout<<"Από παραγόμενη κλάση για τη fun1...\n";}
void Kid_class::fun2()
    {cout<<"Από παραγόμενη κλάση για τη fun2...\n";}

```

- ❖ Προσοχή, ναι μεν η `fun1` είναι η εικονική που έχει ορισθεί στη βασική κλάση, αλλά η `fun2` είναι εδώ απλή συνωνυμία με εκείνη της βασικής κλάσης `Base_class`.

# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (4/5)

```
#include <iostream.h>
void main()
{
    Kid_class kid_object;
    Kid_class *pkid = &kid_object;
    Base_class *pbase = &kid_object;
    pbase->fun1(); // κλήση εικονικής
    pbase->fun2(); // κλήση μη εικονικής
    pkid->fun1(); // κλήση εικονικής από παραγώμενη
    pkid->fun2(); // κλήση τοπικής, μη εικονικής
}
```



# ΕΙΚΟΝΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ (5/5)

- ❖ Η Κλήση μιας εικονικής, ανεξάρτητα από το αν γίνει μέσα από αντικείμενα της βασικής ή της παραγόμενης κλάσης, θα εκτελεσθεί τελικά το σώμα της συνάρτησης αυτής που βρίσκεται μέσα στην παραγόμενη κλάση.

Από παραγόμενη κλάση για τη fun1...

Από βασική κλάση για τη fun2...

Από παραγόμενη κλάση για τη fun1...

Από παραγόμενη κλάση για τη fun2...

- ❖ In object-oriented programming when a derived class inherits from a base class, **an object of the derived class may be referred to via a pointer or reference of either the base class type or the derived class type.** If there are base class methods overridden by the derived class, the method actually called by such a reference or pointer can be bound either 'early' (by the compiler), according to the declared type of the pointer or reference, or 'late' (i.e. by the runtime system of the language), according to the actual type of the object referred to.
- ❖ Virtual functions are resolved 'late'. If the function in question is 'virtual' in the base class, the most-derived class's implementation of the function is called according to the actual type of the object referred to, regardless of the declared type of the pointer or reference. If it is not 'virtual', the method is resolved 'early' and the function called is selected according to the declared type of the pointer or reference.

# ΤΕΛΕΣΤΕΣ ΡΗΤΗΣ ΜΕΤΑΤΡΟΠΗΣ (1/5)

❖ Μέσα στα πλαίσια του πολυμορφισμού δημιουργούνται ιεραρχίες κλάσεων, όπου ένα αντικείμενο πρέπει να έχει τη δυνατότητα μετατροπής του από τη μια κλάση στην άλλη. Η δυνατότητα αυτή παρέχεται με τον τελεστή ρητής μετατροπής, ο οποίος εμφανίζεται με τέσσερις μορφές:

- (i) **dynamic\_cast** Για μετατροπές μέσα σε πολυμορφικά φαινόμενα κλάσεων με εικονικές συναρτήσεις.
- (ii) **static\_cast** Για μετατροπές σε μη πολυμορφικούς τύπους, απλούς τύπους δεδομένων ή και σύνθετους, χωρίς όμως την ενσωμάτωση κώδικα για έλεγχο στη διάρκεια της εκτέλεσης.
- (iii) **const\_cast** Για μετατροπή σταθερών τιμών, όπου στην ουσία μια σταθερή τιμή μετατρέπεται σε σταθερά άλλου τύπου.
- (iv) **reinterpret\_cast** Για μετατροπή απλών τιμών, χωρίς κανένα έλεγχο, ούτε κατά τη διάρκεια του compilation.

# ΤΕΛΕΣΤΕΣ ΡΗΤΗΣ ΜΕΤΑΤΡΟΠΗΣ (2/5)

❖ Γενικός συντακτικός τύπος:

```
x_cast<new_value>(old_value) ;
```

❖ όπου x μια από τις λέξεις κλειδιά dynamic, static const και reinterpret, που σε κάθε περίπτωση το old\_value γίνεται τύπου new\_value.

# ΤΕΛΕΣΤΕΣ ΡΗΤΗΣ ΜΕΤΑΤΡΟΠΗΣ (3/5)

❖ Ο τελεστής **dynamic\_cast** χρησιμοποιείται σε πολυμορφισμούς για **ανιούσες (upcast)** ή και **κατιούσες (downcast)** μετατροπές.

❖ Παράδειγμα: Αν η B είναι παραγόμενη κλάση της A, το pb μετατρέπεται με **upcast** σε αντικείμενο της κλάσης A

```
class Base { //...εντολές της βασικής Base...};
class After:public Base{ //εντολές της After};
void fun()
{
    Base p1 = new After;
    Base p2 = new Base;
    After *pa1 = dynamic_cast<After*>(p1);
    After *pa2 = dynamic_cast<After*>(p2);
    //Λάθος, τελικά θα ισχύει pa2 == NULL
    .....
}
```

❖ Μια κατιούσα μετατροπή ισχύει μόνο αν προηγουμένως έχει προσδιορισθεί ότι το αντικείμενο ανήκει σε μια υψηλά ιστάμενη ιεραρχικά κλάση, αλλά παίρνει τιμές μόνο σε μια παραγόμενη κλάση, οπότε έχει νόημα η ρητή μετατροπή.

# ΤΕΛΕΣΤΕΣ ΡΗΤΗΣ ΜΕΤΑΤΡΟΠΗΣ (4/5)

- ❖ Ο τελεστής **static\_cast** πραγματοποιεί και αυτός **αλλαγή κλάσης** προς την αρχή ή το τέλος της ιεραρχίας αντικειμένων μέσα σε πολυμορφικές δομές, **χωρίς όμως έλεγχο εφικτότητας**.
  - ◊ Οι ανιούσες αλλαγές θεωρούνται ασφαλείς, ενώ κάθε τι άλλο αφήνεται στη λογική του χρήστη.
- ❖ Ο **static\_cast** χρησιμοποιείται επίσης για **μετατροπές απλών δεδομένων**, εφόσον ο χρήστης γνωρίζει το πλήθος των bytes που απαιτεί το αριστερό μέλος της μετατροπής, το οποίο πρέπει να υπερκαλύπτει τα bytes που απαιτούνται για το δεξιό μέλος.

# ΤΕΛΕΣΤΕΣ ΡΗΤΗΣ ΜΕΤΑΤΡΟΠΗΣ (5/5)

## ❖ Παράδειγμα μετατροπής απλών δεδομένων

.....

```
char ch;
```

```
float f1;
```

```
double d1;
```

```
int it = 68;
```

.....

```
i = static_cast<int>(ch);
```

```
//από χαρακτήρα σε ακέραιο
```

```
d1 = static_cast<double>(f1);
```

```
//από float σε double
```

.....